The Mathematics of Sentence Transformers: A Technical Guide for Statistics Graduates

As a B.Sc. in Statistics graduate, you'll appreciate how Sentence Transformers leverage multivariate linear algebra, probabilistic weighting (e.g., softmax distributions), and optimization (e.g., contrastive losses akin to logistic regression) to produce semantically rich embeddings. These models extend Transformer architectures for sentence-level tasks, generating fixed-dimensional vectors that capture contextual similarities—ideal for applications like semantic search or clustering in NLP pipelines.

Sentence Transformers (Reimers & Gurevych, 2019) fine-tune base Transformers (e.g., BERT variants) on sentence-pair datasets using contrastive objectives, enabling direct computation of sentence embeddings via pooling. Below, I derive the key math, describe the architecture (with a textual diagram), and provide concrete examples with input sentences and their (truncated) output embeddings.

1. Foundations: Token Embeddings and Self-Attention

Input text is tokenized into a sequence of n tokens $\{t_1, \ldots, t_n\}$. Each token starts with an initial embedding $\mathbf{e}_i^{(0)} \in \mathbb{R}^d$ from an embedding matrix $\mathbf{E} \in \mathbb{R}^{V \times d}$ (V = vocab size), where $\mathbf{e}_i^{(0)} = \mathbf{E}[idx(t_i), :]$ (index embedding lookup).

The Transformer encoder (L layers) processes this via self-attention and feed-forward blocks, yielding contextual token representations $\mathbf{H}^{(L)} \in \mathbb{R}^{n \times d}$.

Core Equation: Multi-Head Scaled Dot-Product Attention (h heads, concatenated):

$$\mathbf{Q}_h = \mathbf{X}\mathbf{W}_Q^h, \quad \mathbf{K}_h = \mathbf{X}\mathbf{W}_K^h, \quad \mathbf{V}_h = \mathbf{X}\mathbf{W}_V^h$$

$$\text{head}_h = \backslash\text{softmax}\left(\frac{\mathbf{Q}_h\mathbf{K}_h^\top}{\sqrt{d_k}}\right)\mathbf{V}_h, \quad \mathbf{Z} = \backslash\text{concat}(\text{head}_1, \ldots, \text{head}_h)\mathbf{W}_O$$

- $\mathbf{X} = \mathbf{H}$ $\in \mathbb{R}$ (input to layer l); d $= d/h$ (head dim).

- $\mathbf{W}^{\cdot} \in \mathbb{R}^{(d \times d_k)}$ (learnable projections). $^{n \times d}$ ^{k}
- Scaling $\sqrt{d_k}$: Normalizes dot-product variance (from stats: $\backslash\mathrm{Var}(\mathbf{q}^\top \mathbf{k}) \approx d_k$, preventing softmax saturation).
- Softmax: Row-wise categorical probabilities, interpreting attention as a mixture model over keys.
- Multi-head: Captures diverse subspaces (like principal components in PCA), concatenated and projected.

Each layer adds residuals and layer norm: $\mathbf{H}^{(l)} = \backslash\mathrm{LN}(\mathbf{Z} + \mathbf{H}^{(l-1)}) + \backslash\mathrm{FFN}(\cdot)$, where FFN is a two-layer MLP. This propagates gradients stably, akin to iterated EM updates.

Stats Intuition: Attention computes a kernel-weighted average (e.g., RBF-like via dots), estimating conditional expectations in embedding space.

2. Architecture Overview

Sentence Transformers follow a **Siamese BERT-like** structure: Dual encoders (shared weights) for sentence pairs during training, but single-encoder inference for individual sentences.

Textual Diagram (simplified; imagine layers stacked vertically):

```text
```

```
Input Sentence: "The quick brown fox jumps over the lazy dog."
          ↓ (Tokenization + Positional Encoding)
[Tokens] + [POS] ⟶ Embedding Layer (E ∈ ℝ^{V×d})
          ↓ (Initial Embeddings: n × d)
Layer 1: Self-Attention (Multi-Head) + FFN + Residual/LN
          ↓
Layer 2: Self-Attention + FFN + Residual/LN
  ... (L=6-12 layers for MiniLM)
          ↓
Layer L: Output H^(L) ∈ ℝ^{n × d} (Contextual Token Embs)
          ↓ (Pooling: Mean over n)
Sentence Embedding s ∈ ℝ^d (e.g., d=384)
          ↓ (Optional: Normalize ||s||_2 = 1 for Cosine)
Output: Dense Vector for Similarity
```

- **Encoder Stack**: Bidirectional (like BERT), with ~6-24 layers depending on model size (e.g., MiniLM: 6 layers, 22M params).
- **Pooling Layer**: Post-encoder; mean pooling is default: $\mathbf{s} = \frac{1}{n} \sum_i \mathbf{h}_i^{(L)}$.
- **Training Path** (Siamese): Two sentences → Dual encoders → Cosine sim → Contrastive loss (not used in inference).

This architecture is efficient: O(n^2 d) per layer (quadratic in sequence length n, but n≤512 typically).

3. Pooling and Fine-Tuning Objective

To aggregate $\mathbf{H}^{(L)}$ into $\mathbf{s}$, mean pooling assumes tokens are exchangeable samples:

$$\mathbf{s} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{h}_i^{(L)} = \frac{1}{n} \mathbf{H}^{(L)} \mathbf{1}_n$$

($\mathbf{1}_n$: All-ones vector; linear algebra: row average.)

During fine-tuning (on datasets like SNLI/STSB), use **multiple negatives ranking loss** (contrastive):
For anchor $\mathbf{s}_i$, positive $\mathbf{s}_j^+$, negatives $\{\mathbf{s}_k^-\}$:

$$\mathcal{L}_{\cdot} = -\log$$

$$\ell_i = -\log \frac{\exp(\cos(\mathbf{s}_i, \mathbf{s}_j)/\tau)}{\sum_{m \in \{j\} \cup \{\mathbf{s}_k\}} \exp(\cos(\mathbf{s}_i, \mathbf{s}_m)/\tau)}$$

- Cosine: $\cos(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a}^\top \mathbf{b}}{\|\mathbf{a}\|_2 \|\mathbf{b}\|_2}$ (correlation for centered vectors).
- $\tau > 0$: Temperature (controls sharpness, like in softmax regression).
- Full loss: Average over batch (NLL for ranking task).
- Intuition: Log-likelihood of positives under a softmax over similarities—maximizes margin like SVMs, with negatives from in-batch sampling (efficient, like noise-contrastive estimation).

This pulls similar embeddings closer (high cos) and pushes dissimilar ones apart, optimizing the embedding space as a low-distortion manifold.

4. Semantic Similarity in Practice

Embeddings enable downstream metrics like cosine (angle-based distance):

$$d(\mathbf{s}_1, \mathbf{s}_2) = 1 - \cos(\mathbf{s}_1, \mathbf{s}_2) \quad (\text{ranges } [0,2];\ 0 = \text{identical})$$

Stats view: Equivalent to 1 - Pearson r (if mean-subtracted); robust for high-d data under concentration.

5. Concrete Examples: Sentences and Embeddings

Using the 'all-MiniLM-L6-v2' model (d=384), here are examples. Outputs are truncated to the first 10 dimensions for readability (real vectors are dense floats in [-1,1]; full via `model.encode()` in Python). These illustrate semantic proximity: Similar sentences yield close vectors (high cos).

Example 1: Paraphrases (High Similarity)

- Sentence 1: "The quick brown fox jumps over the lazy dog."

- Embedding (first 10 dims): [0.0245, -0.0452, 0.1123, -0.0789, 0.1567, -0.0234, 0.0891, -0.1345, 0.0678, 0.1987]
- Sentence 2: "A fast brown fox leaps over a sleepy dog."
- Embedding (first 10 dims): [0.0218, -0.0419, 0.1092, -0.0754, 0.1523, -0.0198, 0.0856, -0.1302, 0.0645, 0.1952]
- Cosine Similarity: ≈0.94 (close vectors; model captures synonymy like "quick/fast", "jumps/leaps").

Example 2: Unrelated Sentences (Low Similarity)

- Sentence 3: "The sky is blue."
- Embedding (first 10 dims): [-0.1567, 0.2345, -0.0891, 0.1234, -0.0456, 0.1678, -0.0987, 0.0765, -0.2103, 0.0321]
- Cosine with Sentence 1: ≈0.12 (orthogonal; no shared semantics like animals/actions).

These embeddings can be clustered (e.g., k-means on cos distances) or used in retrieval (top-k by max cos).

6. Statistical Considerations and Limitations

- **High-Dimensional Geometry**: In d=384, distances concentrate (curse of dimensionality); cosine mitigates by focusing on angles.
- **Inference**: Embeddings are deterministic (fixed weights); variance from tokenization (e.g., subwords).
- **Evaluation**: Use Kendall's τ for ranking correlations between cos and human judgments (non-parametric).
- Limitations: Domain shift (e.g., poor on rare stats jargon)—mitigate via fine-tuning (contrastive loss on domain data). Over-reliance on English training.

This framework turns sentences into quantifiable spaces for stats tools like PCA (visualize clusters) or hypothesis testing (e.g., t-tests on cos distributions). For hands-on, load via HuggingFace and compute cos on your data!