**1. Embedding Layer**

* **Definition**: The embedding layer is responsible for converting tokenized input into dense vectors of a fixed size, known as embeddings. In text processing, each token (word or sub-word) is mapped to a high-dimensional vector space where similar words have similar vector representations.
* **Input and Output Shape**:
  + **Input**: (Batch=32, Seq\_len=64) → This is a batch of 32 sequences, each of length 64 (64 tokens per sequence).
  + **Output**: (Batch=32, Seq\_len=64, Embed\_dim=16) → The output is a 3D tensor where each token is represented by a 16-dimensional vector.
* **Purpose**: It transforms discrete token IDs into continuous, dense vector representations. The model can now work with these vectors to learn patterns and relationships between words.

**2. Transpose 1**

* **Definition**: A transpose operation switches the axes of a tensor. In this case, we are switching the sequence length (Seq\_len) with the batch size (Batch). This is done to prepare the data for the next layers where sequence processing is more convenient with the sequence length as the leading dimension.
* **Input and Output Shape**:
  + **Input**: (Batch=32, Seq\_len=64, Embed\_dim=16)
  + **Output**: (Seq\_len=64, Batch=32, Embed\_dim=16) → Now, sequence length becomes the first dimension, which makes further operations like positional encoding easier.
* **Purpose**: This operation is a common reshaping step to make sure the tensor is in the right shape for specific operations like attention or positional encoding.

**3. Positional Encoding**

* **Definition**: Positional encoding provides information about the position of each token in the sequence. Since transformer models (and similar architectures like EAM) don’t have an inherent sense of word order, positional encoding is added to the embeddings to retain the positional information of words in a sentence.
* **Input and Output Shape**:
  + **Input**: (Seq\_len=64, Batch=32, Embed\_dim=16)
  + **Output**: (Seq\_len=64, Batch=32, Embed\_dim=16) → The shape remains the same, but the values are modified by adding the positional encodings to each token embedding.
* **Purpose**: This encoding helps the model understand the relative positions of tokens in the sequence, which is crucial for processing sequential data like sentences.

**4. Transpose 2**

* **Definition**: This is another transpose operation that reverses the first transpose. We switch the sequence length back with the batch size. This operation brings the data back into the original format where the batch size is the first dimension.
* **Input and Output Shape**:
  + **Input**: (Seq\_len=64, Batch=32, Embed\_dim=16)
  + **Output**: (Batch=32, Seq\_len=64, Embed\_dim=16) → Now the batch size is the first dimension again, which is required for most neural network layers that expect batch-first input.
* **Purpose**: This ensures the data is back in the batch-first format so it can continue through the model in the expected format.

**5. Token Pooling**

* **Definition**: Token pooling is a down-sampling operation that reduces the sequence length by combining information from adjacent tokens. In this case, token pooling is done with a pool factor of 2, meaning it combines every 2 tokens into one, effectively halving the sequence length.
* **Input and Output Shape**:
  + **Input**: (Batch=32, Seq\_len=64, Embed\_dim=16)
  + **Output**: (Batch=32, Pooled\_Seq\_len=32, Embed\_dim=16) → After pooling, the sequence length is reduced to 32, while the embedding dimension remains the same.
* **Purpose**: Token pooling helps reduce the computational load by reducing the sequence length, which can speed up processing while still retaining important information.

**6. Global Average Pooling**

* **Definition**: Global average pooling is a down-sampling layer that computes the average of each feature (across the sequence) and reduces the sequence to a single vector per input. It collapses the sequence dimension by averaging the values across the sequence length for each feature.
* **Input and Output Shape**:
  + **Input**: (Batch=32, Pooled\_Seq\_len=32, Embed\_dim=16)
  + **Output**: (Batch=32, Embed\_dim=16) → After global average pooling, we end up with a vector of size 16 for each sequence, reducing the sequence dimension to 1.
* **Purpose**: Global average pooling is a common technique to reduce the dimensionality and aggregate information across the entire sequence. It helps create a fixed-size vector representation regardless of the input sequence length.

**7. Fully Connected Layer**

* **Definition**: A fully connected (dense) layer takes the pooled output and maps it to the desired number of output classes (in this case, 10). Each feature (embedding) is connected to each output class through a set of learned weights.
* **Input and Output Shape**:
  + **Input**: (Batch=32, Embed\_dim=16)
  + **Output**: (Batch=32, Num\_Classes=10) → The output is a vector of size 10 for each sequence, representing the class scores for classification.
* **Purpose**: This layer applies a linear transformation to the pooled output and outputs a prediction for each class.

**8. Output Layer**

* **Definition**: The output layer is the final layer that provides the model’s predictions. It contains 10 output neurons, one for each possible class. The final output can be passed through a softmax function to get probabilities for classification tasks.
* **Input and Output Shape**:
  + **Input**: (Batch=32, Num\_Classes=10)
  + **Output**: (Batch=32, Num\_Classes=10) → This is the final output, where each sequence has a vector of 10 values corresponding to the predicted class scores.
* **Purpose**: The output layer provides the final predictions, which can be further processed to get probabilities or final class predictions.



