1. Embedding and positional encoding

Input: The input sequence for each user (User1 and User2) is transformed into an embedding matrix of size [50, 50], where the actual item embeddings are followed by zeros for padding.

Positional encoding: This embedding matrix is then element-wise added with a positional encoding matrix of the same size to incorporate the sequence order information.

2. First normalization layer

Normalization: Before feeding the data into the self-attention blocks, the embedding matrix typically goes through a normalization step, such as Layer Normalization. This helps in stabilizing the learning process by normalizing the data across the features.

Sequence: The normalized matrix for the first sequence would still be [50, 50].

3. Self-Attention block

Self-Attention mechanism: This layer processes the normalized embeddings to model relationships between all items in the sequence. The attention mechanism allows the model to focus differently on each item based on the context provided by other items in the sequence.

Dropout: Dropout is applied within the self-attention mechanism to prevent overfitting by randomly dropping units (features) during training, which helps improve model generalization.

Residual connection: There is typically a residual connection that adds the input of the self-attention block to its output, which helps in mitigating the vanishing gradient problem by allowing gradients to flow directly through the network.

4. Second normalization layer

Post-attention normalization: After the self-attention and residual connection, another layer normalization might occur. This second normalization again stabilizes the training process by ensuring that the output from the self-attention block is normalized before being fed into the subsequent feed-forward network.

5. Feed-Forward network

Network structure: The feed-forward network in Transformer-based models usually consists of two linear (fully connected) layers. Between these layers, a non-linearity such as ReLU is applied.

1D convolutional layers: In some implementations, these layers can be replaced by or understood as 1D convolutional layers where the convolution is applied across the feature dimension. The first layer increases dimensions (expansion), includes ReLU activation, and the second compresses it back (projection), potentially with a linear activation.

Dropout: Dropout applied between these layers to further ensure robustness against overfitting.

Step 6: Second residual connection

Second residual connection: After the output from the feed-forward network, another residual connection is typically applied. This involves adding the output of the feed-forward network to its input (which itself is the output of the previous self-attention block after normalization).

Functionality: This additional residual connection helps to preserve the identity of the information flowing through the network, ensuring that valuable information isn't lost as it passes through deeper layers. It also helps combat the vanishing gradient problem by

providing alternative pathways for gradients during backpropagation, supporting deeper model architectures.

Step 7: Subsequent layer normalization

Normalization post-feed-forward: It's common to apply another layer normalization after the residual connection that follows the feed-forward network. This normalization aims to stabilize the learning process by normalizing the layer outputs, similar to previous normalization steps.

Step 8: Second Self-Attention layer (the same structure as for the first one)

Second pass through Self-Attention: The normalized output from the post-feed-forward residual connection is then fed into a second self-attention layer. This second application of self-attention allows the model to refine its understanding of the relationships between items in the sequence even further. Similar to the first self-attention layer, this layer examines the entire sequence and calculates attention scores that determine how much focus to put on each part of the sequence when predicting the next item. The model can adjust its focus based on additional context it may have gleaned from passing through the previous layers.

Step 9: Preprocessing phase: Positive and negative samples

- ***Positive samples***: For each item in a sequence, the **positive sample** is typically the next item in the sequence. For example:
    - Sequence 1 (User1): Positive samples are Item2 (after Item1), Item3 (after Item2), and Item4 (after Item3).
    - Sequence 2 (User2): Positive samples are Item9 (after Item7) and Item11 (after Item9).
- ***Negative samples****:*
    - Negative samples are items that the user has **not interacted with**. These are chosen randomly from the entire dataset, ensuring they do not overlap with the user's sequence or positive samples.
    - The number of negative samples is typically equal to or greater than the number of positive samples, providing a contrasting set for the model to distinguish between relevant and irrelevant items.

Step 10: Dot product calculations

After processing the sequence embeddings through self-attention layers, we have a contextualized embedding for each position in the sequence (e.g., [50, 50] for max length 50 and embedding size 50). These embeddings are then used for dot product calculations:

1. *Dot product with positive samples*:
    - The embedding of each position in the sequence (e.g., Item1's embedding after self-attention) is **dot-multiplied** with the embedding of its corresponding positive sample (e.g., Item2's embedding).
    - The dot product measures the similarity between the contextualized sequence embedding and the positive sample embedding. A higher dot product indicates higher relevance or similarity.
2. ***Dot product with negative samples:***
    - Similarly, the same sequence embeddings are dot-multiplied with the embeddings of the negative samples (e.g., random items not interacted with by the user).

- This measures the similarity between the sequence embedding and irrelevant items. Ideally, the model should learn to assign low similarity scores (dot products) to these negative samples.

Step 11: Loss function calculation: Binary Cross-Entropy

The output of these dot products (for both positive and negative samples) is then used to compute the **binary cross-entropy loss**. Here's how it works:

1. *Predicted scores:*
   - The dot products serve as the **predicted scores** from the model for how likely each item is to be the next item in the sequence.
   - These scores can be interpreted as logits, which are raw predictions before applying any activation function.
2. *Ground truth labels:*
   - The ground truth labels for positive samples are **1** (indicating relevance).
   - The ground truth labels for negative samples are **0** (indicating irrelevance).
3. *Binary Cross-Entropy (BCE)*:
   - BCE is computed as:

$$\text{BCE Loss} = -\frac{1}{N}\sum [y \cdot \log(\sigma(\hat{y})) + (1-y) \cdot \log(1-\sigma(\hat{y}))]$$

     - y: Ground truth label (1 for positive samples, 0 for negative samples).
     - y^: Predicted score (logit from dot product).
     - σ(y^): Sigmoid activation applied to logits to transform them into probabilities.
4. *What happens internally*:
   - The sigmoid function ensures that the predicted scores are squashed into a probability range of [0, 1].
   - BCE loss penalizes the model heavily for assigning high probabilities to negative samples or low probabilities to positive samples, pushing the model to better distinguish between relevant and irrelevant items.
5. Numerical example:
   We got the following scores:
   - dot(item1, item2) = 2.3 *positive*
   - dot(item2, item3) = 2.1 *positive*
   - dot(item3, item4) = 2.5 *positive*
   - dot(item1, itemX) = 0.5 *negative*
   - dot(item2, itemY) = 0.7 *negative*
   - dot(item3, itemZ) = 0.3 *negative*

   Prepare scores and labels:
   Scores: [2.3, 2.1, 2.5, 0.5, 0.7, 0.3)
   Labels: [1, 1, 1, 0, 0, 0] - Grand Truth labels!

   The **BCE loss** is calculated:

   BCE Loss = - ⅙ x **[**1 x log(σ(**2.3**)) + 1 x log(σ(**2.1**)) + 1 x log(σ(**2.5**)) + 1 x log(1 - σ(**0.5**)) + 1 x log(1 - σ(**0.7**)) + 1 x log(1 - σ(**0.3**)) **]** = **0.537**