Notes on Sentence Transformers

compiled from various sources: D. Gueorguiev, 8/11/2025

# Multi-classification using Bi-Encoders

A Sentence-BERT (SBERT) bi-encoder is a model architecture that encodes two separate text inputs independently into fixed-dimensional vector representations (embeddings). These embeddings can then be compared, typically using cosine similarity, to determine the semantic similarity between the original texts.

Here is a basic example of how to use a pre-trained SBERT bi-encoder for semantic search:

from sentence\_transformers import SentenceTransformer, util

# 1. Load a pre-trained SBERT bi-encoder model

# 'all-MiniLM-L6-v2' is a common and efficient choice

model = SentenceTransformer('all-MiniLM-L6-v2')

# 2. Define the sentences to encode

sentences = [

"The cat sat on the mat.",

"A feline rested on the floor covering.",

"The dog barked loudly.",

"What is the capital of France?"

]

# 3. Encode the sentences into embeddings

# The encode method converts each sentence into a vector

sentence\_embeddings = model.encode(sentences, convert\_to\_tensor=True)

# 4. Perform a semantic search (e.g., find the most similar sentence to a query)

query = "A cat is on a rug."

query\_embedding = model.encode(query, convert\_to\_tensor=True)

# Calculate cosine similarity between the query and all sentences

cosine\_scores = util.cos\_sim(query\_embedding, sentence\_embeddings)[0]

# Find the index of the most similar sentence

most\_similar\_index = cosine\_scores.argmax().item()

most\_similar\_sentence = sentences[most\_similar\_index]

similarity\_score = cosine\_scores[most\_similar\_index].item()

print(f"Query: '{query}'")

print(f"Most similar sentence: '{most\_similar\_sentence}' (Similarity: {similarity\_score:.4f})")

# You can also compare any two sentences directly

sentence1 = "I love eating pizza."

sentence2 = "Pizza is my favorite food."

embedding1 = model.encode(sentence1, convert\_to\_tensor=True)

embedding2 = model.encode(sentence2, convert\_to\_tensor=True)

similarity\_between\_sentences = util.cos\_sim(embedding1, embedding2).item()

print(f"Similarity between '{sentence1}' and '{sentence2}': {similarity\_between\_sentences:.4f}")

## The problem of Multi-class Classification

Given is a training data in the following format

query : a text string

query result : a list of choices; each choice is represented by unique identifier

selection:

choice information:

For each choice we are given a set of features which characterize the choice. Each feature can be either numerical or categorical.

Given a query and a set of alternatives resulting from the query execution we would like to predict which alternative is most likely to be selected for this query.

A common approach involves computing a prediction score for each query-alternative pair:

with the corresponding probabilistic prediction defined as:

choose -th alternative

To train the function f, the cross-entropy loss is typically used:

, represents the score of the actually selected alternative

## Application of Bi-Encoder for the Multi-class classification problem

The simplest architecture for constructing is the Bi-Encoder architecture. Specifically, the prediction score can be constructed as the inner product of the query embedding and the product embedding

# Example Problem: Predicting which Wine the User is most likely to click on

Given a dataset containing clickstream data related to wine purchases. The data includes the user’s search query and the assortment of products presented to them. The objective of this problem is to predict which product the user is most likely to click on.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

# References

[1] [RoBERTa: Robustly Optimized BERT Pretraining Approach, Y. Liu et al, U of Washington, 2019](https://github.com/dimitarpg13/transformer_examples/blob/main/articles/bert/RoBERTa-A_Robustly_Optimized_BERT_Pretraining_Approach_Liu_2019.pdf)

[2] [BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, Jacob Devlin et al, Google AI, 2019](https://github.com/dimitarpg13/transformer_examples/blob/main/articles/bert/BERT-Pre-training_of_Deep_Bidirectional_Transformers_for_Language_Understanding_Devlin_2019.pdf)