NLP 2025 Lab 3 Report: Attention and Pre-trained Models

Marta Adamowska[*] **Katarina Dvornák**[†] **Pavels Sorocenko**[‡]

1 Exercise 1 Unpacking the set

We unpacked the set in the *unpack* function, that iterates through the set and saves full sentences and compressed sentences in the new separate columns.

```
# iterate through the eamples
for set in examples['set']:
    # add the full sentence to
    # the sentences column
    sentences.append(set[0])
    # add the compressed sentence to
    # the compressed_sentences column
    compressed_sentences.append(set[1])
```

2 Exercise 2 Questions about the tokenizer

2.1 What is the size of the vocabulary?

To check the vocabulary size, we accessed the tokenizer object using the *tokenizer.vocab_size* attribute. The output indicates that the vocabulary size is 30,522. The same number is provided in the documentation on Hugging Face (hug).

2.2 What are the special tokens apart from '[CLS]' and '[SEP]'? What are their functions?

According to the documentation on the Hugging Face website (hug), there are three special tokens apart from `[CLS]` and `[SEP]`:

- `[UNK]` — The unknown token; any token not found in the vocabulary is replaced with this one.
- `[PAD]` — Used for padding sequences to a uniform length within a batch, allowing all inputs to be processed together.
- `[MASK]` — The token used for masking values during pretraining. In masked language modeling, the model attempts to predict the original value of these tokens.

3 Exercise 3 Questions about the Model

3.1 What is the number of transformer layers in this model?

By observing that the printed layers start with (0): BertLayer and end with (11): BertLayer, we concludes that the model includes 12 transformer layers. This aligns perfectly with the architecture of BERT-base, which is defined to have **12** encoder layers.

3.2 What is the dimension of the embeddings?

From the model output, we determined the embedding dimension to be **768** using the following line:

```
(word_embeddings): Embedding(30522, 768,
                             padding_idx=0)
```

3.3 What is the hidden size of the FFN in the transformer layer?

According to the lines:

```
(intermediate): BertIntermediate(
    (dense): Linear(in_features=768,
            out_features=3072, bias=True)
   (intermediate_act_fn): GELUActivation()
)
```

the FFN hidden size is **3072**. This value matches the standard BERT-base architecture, where hidden size is equal to 768 and FFN size is equal to $4 \times$ hidden size $= 4 \times 768 = 3072$.

3.4 What is the total number of parameters of the model?

The output of the `num_parameters()` method of the model indicates that it has **109,482,240** parameters.

[*] m.adamowska@student.maastrichtuniversity.nl
[†] k.dvornak@student.maastrichtuniversity.nl
[‡] p.sorocenko@student.maastrichtuniversity.nl

3.5 How can you find the vocabulary size from the model?

The vocabulary size can be determined from the following line of the model architecture: `(word_embeddings): Embedding(30522, 768, padding_idx=0)` The first argument of the `Embedding` layer (i.e., `30522`) represents the number of tokens in the vocabulary. This vocabulary size of **30,522** aligns with the finding from Exercise 2.

4 Exercise 4 Plotting the layer-wise similarities between words

We analyzed how the cosine similarity between specific token representations evolves across the layers of a transformer model. For each layer, we extracted the hidden embeddings corresponding to tokens of interest from pairs of sentences and computed their pairwise cosine similarities.

5 Exercise 5 Evolution of embeddings

5.1 Discuss the plot. Are the plot showing what you expected to see?

In Figure 1, we illustrate the cosine similarity trajectories between three sentences:

1. "We will rob a bank next week!"
2. "The children skipped stones by the bank of the river."
3. "I put money in the bank."

Although all three sentences contain the word "bank", it is used in two distinct semantic contexts—referring to a financial institution in sentences 1 and 3, and to a riverbank in sentence 2. In sentences 1 and 3, the word "bank" shares the same meaning, and as expected, the cosine similarity between these two remains consistently high across all layers. In contrast, the similarity between sentences 1–2 and 2–3 decreases rapidly as the number of layers increase, which is anticipated given that sentence 2 uses "bank" in the context of a riverbank. This behavior indicates the model's growing ability to distinguish word meaning in different context and it demonstrates that deeper layers of the model capture semantic nuances more effectively.

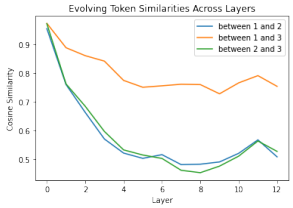

Figure 1: Plot comparing cosine similarities between 3 sentences: 1. "We will rob a bank next week!" 2. "The children skipped stones by the bank of the river." 3. "I put money in the bank."

5.2 Take a look at the similarity between the word "bank" in the first example for the first layer (layer index 0)? This corresponds to the embedding layer of BERT. It is close to '1' but not exactly. Why is that?

The similarity is not exactly 1 because BERT incorporates positional embeddings. The word "bank" appears at different positions across the three sentences - position 5 in text 1, position 7 in text 2 and position 6 in text 3. Although the word itself is identical in all three examples, the cosine similarity at the first layer is less than 1 due to these positional differences. The initial similarity between text 1 and text 2 is the lowest because the position difference for the token of interest is 2. For the other two pairs, the difference is only 1, resulting in a slightly higher starting cosine similarity — though still below 1.

5.3 Plot and analyze the similarities between words "nice", "bad", and "lovely" in the sentences "The weather is nice today.", "The weather is bad today.", and "The weather is lovely today.". Comment on the results. Are the plots showing what you expected to see?

Figure 2 illustrates similarities between words "nice", "bad", and "lovely" in sentences:

1. "The weather is nice today."
2. "The weather is bad today."
3. "The weather is lovely today."

As expected, the cosine similarity between sentences 1 and 3 remains relatively high across layers, as the words "nice" and "lovely" both express a positive sentiment and are often used in similar contexts. On the contrary, the similarity between

sentence pairs 1–2 and 2–3 declines progressively with increasing depth. This behavior aligns with intuition, although the sentences differ by only a single word, the semantic content shifts due to different sentiment of target words. The model captures this distinction in its deeper layers.

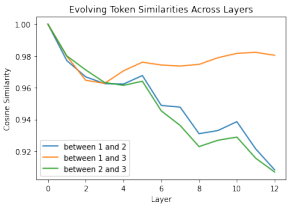

Figure 2: Plot comparing cosine similarities between 3 sentences: 1. "The weather is nice today." 2. "The weather is bad today." 3. "The weather is lovely today."

5.4 Try a different set of sentences and comment on the results.

As a different set we choose the following sentences with a word "light" as a token of interest:

1. "She turned on the light." - "light" in terms of illumination
2. "The box was very light." - "light" in terms of weight
3. "They painted the room a light blue." - "light" in terms of color

Although all three sentences include the same token of interest "light", its meaning varies significantly depending on context. This difference is reflected in the decreasing cosine similarity values across the sentence pairs, as shown in Figure 3. Furthermore, we observed that in sentences 1 and 2, the word "light" appears at the same token position (position 5), which explains why their similarity at layer 0 is equal to 1. In sentence 3, "light" appears at position 6, resulting in a slightly lower initial similarity in comparisons involving this sentence (pairs 1–3 and 2–3).

6 Exercise 6: Implement Sentence Embeddings

In this exercise, we implemented `calculate_sentence_embeddings`, a function that computes sentence embeddings by averaging token representations (hidden states) from a selected BERT layer. We use the attention mask to ignore padded tokens and apply mean

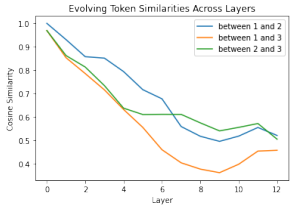

Figure 3: Plot comparing cosine similarities between 3 sentences: 1. "She turned on the light." 2. "The box was very light." 3. "They painted the room a light blue."

pooling over the sequence length, producing a fixed-size embedding of shape `[batch_size, embedding_dim]` (e.g., `[1, 768]` for BERT). By default, the function uses the last hidden layer but allows selecting any specific layer.

Initially, we implemented a basic version of the function that used all tokens (including special tokens like `[CLS]` and `[SEP]`). Later, we updated it to exclude these special tokens, after realizing they introduced noise that impacted similarity results in downstream tasks. To confirm the improvement, we ran a targeted similarity comparison on a set of sentence pairs and found that the updated version yielded more consistent and semantically accurate results.

Final Implementation

```
input_ids = input_batch['input_ids']
attention_mask = input_batch['attention_mask']
hidden_states = model_output['hidden_states'][layer]

# Remove special tokens [CLS] (101), [SEP] (102)
special_tokens_mask = ((input_ids != 101) &
(input_ids != 102)).unsqueeze(-1)

# Combine with attention mask to ignore padding
and special tokens
effective_mask = attention_mask.unsqueeze(-1).float()
special_tokens_mask.float()

# Mask hidden states
masked_hidden_states = hidden_states * effective_mask

# Mean pooling over valid tokens
sum_hidden = masked_hidden_states.sum(dim=1)
lengths = effective_mask.sum(dim=1).clamp(min=1e-9)
sentence_embeddings = sum_hidden / lengths
```

Evaluation and Interpretation

We tested the function using a set of three sentences containing the word *bank* in different con-

texts (crime vs. finance), and visualized the cosine similarity of sentence embeddings across layers. The results showed that:

- Sentences 1 and 3 (both describing a crime) became more similar in deeper layers.
- Sentence 2 (focused on banking and savings) became less similar to the others.

This confirms that deeper layers of BERT capture richer semantic context and can disambiguate polysemous words. Moreover, our controlled similarity tests demonstrated that excluding `[CLS]` and `[SEP]` improves alignment with human expectations and reduces noise from non-content tokens.

7 Exercise 7 Try different sentences

In this exercise we experimented with how sentence embeddings evolve across different layers using two different sets of sentences. Our goal was to test wether the model captures meaning based on context (semantics) or just surface form (lexical overlap). We visualized sentence similarity using cosine similarity between sentence embeddings at each layer.

Set 1: High Lexical Overlap, Different Meanings

Sentences:

- The ball was caught by the player on the field.
- The ball was held in a grand ballroom.
- The ball rolled down the hill and into the street.

We chose this set of sentences because all three sentences share the word "ball" but its meaning differs: it refers to a sports object, a formal event, and a rolling object. This tests the model's ability to distinguish between different meanings of a word.

As expected the figure shows high similarity in early layers, decreasing in deeper layers as the model distinguishes the context. This confirms that the model can capture contextual differences.

Set 2: High Contextual Overlap, Different Words

Sentences:

- A man is driving a car through the city.
- Someone is piloting a vehicle in an urban area.

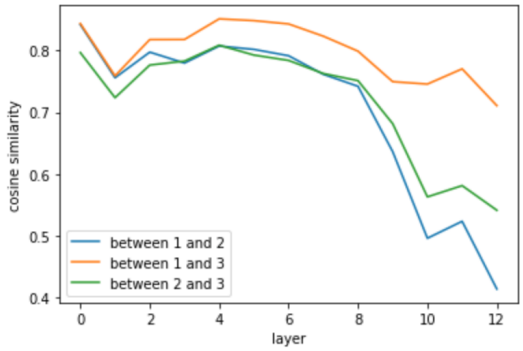

Figure 4: Cosine similarity across layers for sentences with shared lexical tokens but different meanings.

- He navigates traffic downtown in his sedan.

This set was chosen to test if the model can capture shared semantic meaning across paraphrased sentences that use different words.

As shown in the figure, similarity is low in early layers and increases in mid-layers as semantic meaning is extracted. It peaks around layers 6-9, before a noticeable drop. Even though we did not expect it, it can be explained by the fact that the final layers of BERT often encode task-specific representations rather than general semantics. Additionally, sentences 1 and 2 consistently showed higher similarity than the other pairs. This can be caused by their similar syntactic structures ans similar word pairs. While sentence 3 is contextually related it uses more distinct phrases, resulting in slightly lower similarity.

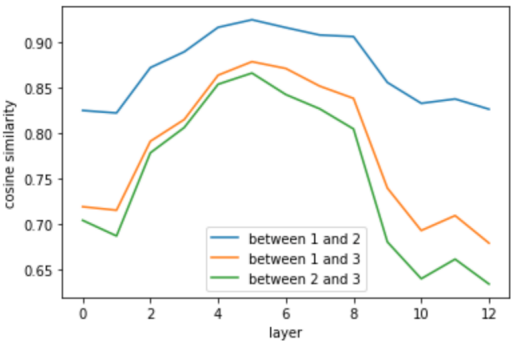

Figure 5: Cosine similarity across layers for sentences with high contextual overlap and varied vocabulary.

This experiment shows that earlier BERT layers reflect lexical overlap, while deeper layers capture semantic meaning. In set 1, similarity decreases across the layers as word meanings become distinguished. In set 2, similarity increases as the model recognizes shared semantics, despite different vo-

cabulary.

8 Exercise 8 Tokenize sentence and compressed

In this exercise we prepared the dataset for sentence embedding by separately tokenizing the sentence and compressed columns. Below are the two tokenization functions:

```
def tokenize_sentence(examples):
    tokenized_sentence = tokenizer(
        examples["sentence"],
        truncation=True,
        return_attention_mask=True,
        return_tensors=None
    )
    return tokenized_sentence

def tokenize_compressed(examples):
    tokenized_compressed = tokenizer(
        examples["compressed"],
        truncation=True,
        return_attention_mask=True,
        return_tensors=None
    )
    return tokenized_compressed
```

Both functions apply `truncation=True` to ensure that the sentences do not exceed the model's maximum input length. They return the `attention_mask` , which is needed later for masking during mean pooling. Additionally, they set `return_tensors=None` to keep outputs compatible with Hugging Face's map() method.

9 Exercise 9 Embed query function

In this exercise we implemented the function `embed_query` to generate a vector representation for a single query. The embedding process follows these steps: tokenizing the input, running it through the model, and applying our `calculate_sentence_embeddings` function to extract a mean-pooled representation.

```
def embed_query(query,
    sentence_embedding_fn):

    query_tokenized = tokenizer(
        query,
        return_tensors="pt",
        truncation=True
    ).to(model.device)

    model_output = model(
        **query_tokenized,
        output_hidden_states=True
    )

    query_embedding = sentence_embedding_fn(
        query_tokenized, model_output
    )

    return query_embedding.detach().cpu()
```

We applied the function to the query `"volcano erupted"` and the result was a 768-dimensional embedding, which is the standard output size for BERT models.

10 Exercise 10 Cosine similarity 1 to n in PyTorch

We implemented the `cosine_similarity_1_to_n` function, which computes the cosine similarity between a single vector (usually a query) and a batch of vectors (e.g, all sentence embeddings). Unlike the NumPy implementation used in previous labs, here we re-implemented the logic using PyTorch. The result is a 1D tensor of size N, where each value representssimilarity to a sentence

```
vector_norm = F.normalize(vector, dim=0)# (D,)
 other_vectors_norm = F.normalize
     (other_vectors, dim=1)  # (N, D)
similarity = torch.matmul(other_vectors_norm,
     vector_norm)  # (N,)
```

WE computed cosine similarity between the embedded query and all sentence embeddings in the dataset. To identify the most relevant sentence, we selected the index with highest similarity value using `np.argmax`, which returned "Small quake shakes central Oklahoma." To extend this even further, we used a function to retrieve top-k most similar sentences. We extracted the 10 most similar sentences to the query and printed their content along with similarity scores.

11 Exercise 11 Experiment with different queries

To evaluate the effectiveness of our BERT-based retrieval system, we designed a structured experiment based on specific linguistic and retrieval challenges. We reused queries from the previous lab to compare BERT's behavior and capabilities. We grouped the

queries into categories targeting common weaknesses in traditional models: synonymy, word order, spelling errors, out-of-vocabulary (OOV) terms, paraphrasing, and negation

11.1 1. Synonymy — "child abducted" vs "child kidnapped"

Both queries returned fairly similar results. Unlike TF-IDF, which treated these as unrelated due to lexical mismatch, BERT recognized their semantic similarity and retrieved articles involving children, danger, or law enforcement. While the similarity scores weren't extremely high, the top results were contextually appropriate. This indicates that BERT can partially capture synonymy through contextualized embeddings.

11.2 2. Word Order Sensitivity — "shoots suspect" vs "suspect shoots"

In TF-IDF and BoW, these queries gave nearly identical results. BERT, however, returned slightly different top matches, with "suspect shoots" retrieving more crime-specific headlines, including direct mentions of shootings and suspects. This shows that BERT is somewhat sensitive to word order and role reversal, likely due to its attention mechanisms.

3. Spelling Robustness — "house floded" (misspelled "flooded") The BERT model failed to return relevant results. The misspelling caused the tokenizer to produce meaningless subwords, and as a result, the embeddings were uninformative. This reflects a limitation of BERT-based systems.

4. Out-of-Vocabulary — "ebola" While TF-IDF returned zero-similarity results due to unseen vocabulary, BERT's subword tokenizer allowed it to interpret the query. It surfaced semantically related results about diseases (e.g., HIV, health, vaccinations). This shows BERT's ability to generalize even to rare or previously unseen terms.

5. Paraphrasing — "female dies" vs "woman killed" TF-IDF handled these queries very differently due to lexical mismatch. BERT, however, gave reasonably similar and semantically consistent responses to both. Results included headlines related to death, violence, or gender-specific victims. The similarity gap between the two queries was smaller than in TF-IDF.

6. Negation — "storm hit" vs "storm did not hit" Negation is a well-known weakness for non-contextual models. While BERT's results were not perfect, the model showed a noticeable difference in retrieved headlines — "storm did not hit" returned reports of warnings, forecast changes, or situations with no damage, while "storm hit" returned impact-heavy reports. This illustrates BERT's limited but better sensitivity to negation compared to TF-IDF and BoW.

These findings confirm that BERT embeddings capture semantic meaning and context far more effectively than traditional vector-space models, making them much better suited for sentence-level search and retrieval tasks.

12 Exercise 12 Cosine similarity m to n in PyTorch

We implemented a function *cosine_similarity_m_to_n*, that calculates the cosine similarity between a multiple vectors and other vectors. It uses PyTorch library to first normalize all vectors and then compute matrix product of two tensors to obtain the similarity scores. The implementation is as follows:

```
# normalize both sets of vectors
vectors_norm = F.normalize(vectors, p=2, dim=1)
other_vectors_norm = F.normalize(other_vectors, p=2, dim=1)
# compute cosine similarity as dot product of normalized vectors
similarity = torch.matmul(vectors_norm, other_vectors_norm.T)

return similarity
```

13 Exercise 13 Recall for different k-s

We calculated the recall for $K = 3, 5, 10, 15, 50, 100$. The recall values for different values of K increase consistently as the value of K grows. At $K = 1$, the model retrieves the correct sentence 61.28% of the time, indicating strong top-1 accuracy. Recall rises to 78.50% at $K = 5$ and 86.26% at $K = 15$, eventually reaching 95.22% at $K = 100$. Obtained results are expected, as larger K allows more candidate sentences to be considered, so the probability of retrieving the correct one increases. Overall, the findings suggest that the model performs well, with recall improving steadily as more retrieval candidates are included.

14 Exercise 14 Different ways of embedding sentences

The chosen way of embedding sentences differently was averaging n last layers of BERT, since it promised to capture a deeper semantic meaning. The results were following: Compared to the old functions (one that took into consideration [CLS] and the other that did not), the last layer averaging did increase the cosine similarity of queries

to corpus, with the increasing number of layers. The significant increase in similarity (0.15-0.22 points), taken the same queries as were used on different embedding method, was observed in similar phrases and phrases that negate themselves (e.g. 'police officer' ≅ 'firefighter' and 'storm hits' ≅ 'storm did not hit'). The conclusion may be the fact that indeed the layer averaging can capture a better semantic meaning.

15 Exercise 15 Different model

For the different model MSMARCO has been chosen. The representation of tokens did not differ a lot. Noticeable difference was that it did not separate word endings and stem of theirs often. But when comparing the sentence retrieval it becomes evident that model MSMARCO performed worse than BERT. The similarity of each sentence retrieved for all categories were from 0.43 to 0.57, whereas BERT captured up to 0.80 similarity. Although MSMARCO performed worse, the retrieved sentences were less matching not by a big margin.

16 Exercise 16 Comparison between models

It can be said that in the retrieval task MSMARCO struggles in comparison to BERT, the latter successfully retrieves a necessary headline whereas MSMARCO fails periodically. It still outperformes TF-IDF and BOW methods, since it can work with different word order, it can capture the meaning of rare words. Additionally, negation and synonymy-wise MSMARCO outperformed BERT. Overall, MSMARCO adds a bigger similarity score to any 2 sentences than BERT does, which is a vague sign of overfitting. A good example is:

1.'the building collapsed' ≅ 'structure fell down': similarityBERT = 0.7638 – similarityMARCO = 0.8847
2.'the building collapsed' ≅ 'the dog barked loudly': similarityBERT = 0.5650 – similarityMARCO = 0.8382
3.'child abducted' ≅ 'child kidnapped': similarityBERT = 0.9423 – similarityMARCO = 0.9743
4.'police officer' ≅ 'firefighter': similarityBERT = 0.6085 – similarityMARCO = 0.8652
5.'storm hits' ≅ 'storm did not hit': similarityBERT = 0.6874 – similarityMARCO = 0.9134

Second line shows the flaw brightly. On the other hand, when multiple layers are being averaged during the embedding session, the problem disappears and MSMARCO starts behaving slightly worse than BERT. It may be caused because averaging several last layers of the transformer captures semantics better than a whole new model, being a more robust method than transformer.

References

BERT — huggingface.co. https://huggingface.co/docs/transformers/model_doc/bert#transformers.BertTokenizerFast. [Accessed 29-04-2025].

A Collaborators outside our group

B Use of genAI

Use of genAI tools (e.g. chatGPT), websites (e.g. stackoverflow): list websites where you found code (or other info) as well as include information on how you used genAI tools (e.g. prompts). Failure to disclose use of genAI tools is violation of adademic integrity.