Deep Learning School

Глубокое обучение. Часть 2

Домашнее задание по теме "Механизм внимания"

Это домашнее задание проходит в формате peer-review. Это означает, что его будут проверять ваши однокурсники. Поэтому пишите разборчивый код, добавляйте комментарии и пишите выводы после проделанной работы.

В этом задании вы будете решать задачу классификации математических задач по темам (многоклассовая классификация) с помощью Transformer.

В качестве датасета возьмем датасет математических задач по разным темам. Нам необходим следующий файл:

[Файл с классами](https://docs.google.com/spreadsheets/d/13YIbphbWc62sfa-bCh8MLQWKizaXbQK9/edit?usp=drive_link&ouid=104379615679964018037&rtpof=true&sd=true)

Hint: не перезаписывайте модели, которые вы получите на каждом из этапов этого дз. Они ещё понадобятся.

Импортируем библиотеки, необходимые для работы

In [1]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
import copy
import gc
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import random
import string
import seaborn as sns
sns.set(palette='summer')

from sklearn.model_selection import train_test_split
from tqdm.auto import tqdm

import nltk
import re
nltk.download('stopwords')
from nltk.corpus import stopwords

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, Dataset
from torch.utils.tensorboard import SummaryWriter

import transformers
from transformers import BertTokenizer, BertModel, get_scheduler
```

```
[nltk_data] Downloading package stopwords to /usr/share/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [7]:
```
%pip install evaluate
!pip install bertviz > l.txt
from bertviz import head_view, model_view
```

```
Requirement already satisfied: evaluate in /opt/conda/lib/python3.10/site-packages (0.4.3)
Requirement already satisfied: datasets>=2.0.0 in /opt/conda/lib/python3.10/site-packages (from evaluate) (3.0.1)
Requirement already satisfied: numpy>=1.17 in /opt/conda/lib/python3.10/site-packages (from evaluate) (1.26.4)
Requirement already satisfied: dill in /opt/conda/lib/python3.10/site-packages (from evaluate) (0.3.8)
Requirement already satisfied: pandas in /opt/conda/lib/python3.10/site-packages (from evaluate) (2.2.2)
Requirement already satisfied: requests>=2.19.0 in /opt/conda/lib/python3.10/site-packages (from evaluate) (2.32.
3)
Requirement already satisfied: tqdm>=4.62.1 in /opt/conda/lib/python3.10/site-packages (from evaluate) (4.66.4)
Requirement already satisfied: xxhash in /opt/conda/lib/python3.10/site-packages (from evaluate) (3.4.1)
Requirement already satisfied: multiprocess in /opt/conda/lib/python3.10/site-packages (from evaluate) (0.70.16)
Requirement already satisfied: fsspec>=2021.05.0 in /opt/conda/lib/python3.10/site-packages (from fsspec[http]>=20
21.05.0->evaluate) (2024.6.1)
Requirement already satisfied: huggingface-hub>=0.7.0 in /opt/conda/lib/python3.10/site-packages (from evaluate)
(0.25.1)
Requirement already satisfied: packaging in /opt/conda/lib/python3.10/site-packages (from evaluate) (21.3)
Requirement already satisfied: filelock in /opt/conda/lib/python3.10/site-packages (from datasets>=2.0.0->evaluat
e) (3.15.1)
Requirement already satisfied: pyarrow>=15.0.0 in /opt/conda/lib/python3.10/site-packages (from datasets>=2.0.0->e
valuate) (16.1.0)
Requirement already satisfied: aiohttp in /opt/conda/lib/python3.10/site-packages (from datasets>=2.0.0->evaluate)
(3.9.5)
Requirement already satisfied: pyyaml>=5.1 in /opt/conda/lib/python3.10/site-packages (from datasets>=2.0.0->evalu
ate) (6.0.2)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /opt/conda/lib/python3.10/site-packages (from hugging
face-hub>=0.7.0->evaluate) (4.12.2)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /opt/conda/lib/python3.10/site-packages (from packaging
->evaluate) (3.1.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /opt/conda/lib/python3.10/site-packages (from requests>
=2.19.0->evaluate) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.10/site-packages (from requests>=2.19.0->eva
luate) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /opt/conda/lib/python3.10/site-packages (from requests>=2.19.
0->evaluate) (1.26.18)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/python3.10/site-packages (from requests>=2.19.
0->evaluate) (2024.8.30)
Requirement already satisfied: python-dateutil>=2.8.2 in /opt/conda/lib/python3.10/site-packages (from pandas->eva
luate) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /opt/conda/lib/python3.10/site-packages (from pandas->evaluate) (20
24.1)
Requirement already satisfied: tzdata>=2022.7 in /opt/conda/lib/python3.10/site-packages (from pandas->evaluate)
(2024.1)
Requirement already satisfied: aiosignal>=1.1.2 in /opt/conda/lib/python3.10/site-packages (from aiohttp->datasets
>=2.0.0->evaluate) (1.3.1)
Requirement already satisfied: attrs>=17.3.0 in /opt/conda/lib/python3.10/site-packages (from aiohttp->datasets>=
2.0.0->evaluate) (23.2.0)
Requirement already satisfied: frozenlist>=1.1.1 in /opt/conda/lib/python3.10/site-packages (from aiohttp->dataset
s>=2.0.0->evaluate) (1.4.1)
Requirement already satisfied: multidict<7.0,>=4.5 in /opt/conda/lib/python3.10/site-packages (from aiohttp->datas
ets>=2.0.0->evaluate) (6.0.5)
Requirement already satisfied: yarl<2.0,>=1.0 in /opt/conda/lib/python3.10/site-packages (from aiohttp->datasets>=
2.0.0->evaluate) (1.9.4)
Requirement already satisfied: async-timeout<5.0,>=4.0 in /opt/conda/lib/python3.10/site-packages (from aiohttp->d
atasets>=2.0.0->evaluate) (4.0.3)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.10/site-packages (from python-dateutil>=2.8.2->p
andas->evaluate) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

In [8]:
```
!pip install openpyxl
```

```
Requirement already satisfied: openpyxl in /opt/conda/lib/python3.10/site-packages (3.1.5)
Requirement already satisfied: et-xmlfile in /opt/conda/lib/python3.10/site-packages (from openpyxl) (1.1.0)
```

In [9]:
```
!pip install pymorphy2
```

```
Collecting pymorphy2
  Downloading pymorphy2-0.9.1-py3-none-any.whl.metadata (3.6 kB)
Collecting dawg-python>=0.7.1 (from pymorphy2)
  Downloading DAWG_Python-0.7.2-py2.py3-none-any.whl.metadata (7.0 kB)
Collecting pymorphy2-dicts-ru<3.0,>=2.4 (from pymorphy2)
  Downloading pymorphy2_dicts_ru-2.4.417127.4579844-py2.py3-none-any.whl.metadata (2.1 kB)
Requirement already satisfied: docopt>=0.6 in /opt/conda/lib/python3.10/site-packages (from pymorphy2) (0.6.2)
Downloading pymorphy2-0.9.1-py3-none-any.whl (55 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 55.5/55.5 kB 2.3 MB/s eta 0:00:00
Downloading DAWG_Python-0.7.2-py2.py3-none-any.whl (11 kB)
Downloading pymorphy2_dicts_ru-2.4.417127.4579844-py2.py3-none-any.whl (8.2 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.2/8.2 MB 83.5 MB/s eta 0:00:00:00:010:01
Installing collected packages: pymorphy2-dicts-ru, dawg-python, pymorphy2
Successfully installed dawg-python-0.7.2 pymorphy2-0.9.1 pymorphy2-dicts-ru-2.4.417127.4579844
```

In [10]:
```
device = 'cuda' if torch.cuda.is_available() else 'cpu'
device
```

Out[10]:
```
'cuda'
```

In [11]:
```
SEED = 54

def fixedseed (seed: int=54):
  random.seed(seed)
  os.environ['PYTHONHASHSEED'] = str(seed)
  np.random.seed(seed)
  torch.manual_seed(seed)
  torch.backends.cudnn.benchmark = False
  torch.backends.cudnn.deterministic = True

fixedseed(SEED)
```

Исследование и предобработка данных

In [12]:
```
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
/kaggle/input/data-for-transformer-hw/data_problems_translated.xlsx
```

In [13]:
```
file_path = '/kaggle/input/data-for-transformer-hw/data_problems_translated.xlsx'
data = pd.read_excel(file_path)
data = data.drop(columns='Unnamed: 0')
print(data.shape)
data.head()
```

```
(5273, 2)
```

Out[13]:

	problem_text	topic
0	To prove that the sum of the numbers of the ex...	number_theory
1	(b) Will the statement of the previous challe...	number_theory
2	The quadratic three-member graph with the coef...	polynoms
3	Can you draw on the surface of Rubik's cube a ...	combinatorics
4	Dima, who came from Vrunlandia, said that ther...	graphs

In [14]:
```
print(f'Количество уникальных topics: {data.topic.nunique()}')
data.topic.value_counts()
```

```
Количество уникальных topics: 7
```

Out[14]:
```
topic
number_theory    2396
combinatorics    1020
dirichlet         441
polynoms          426
graphs            384
geometry          371
invariant         235
Name: count, dtype: int64
```

In [15]:
```
data.shape
```

Out[15]:
```
(5273, 2)
```

In [16]:
```
data[data.duplicated()].shape[0]
```

Out[16]:
```
29
```

In [17]:
```
df1 = data.apply(lambda x:sum(x.duplicated()))
print(df1)
```

```
problem_text     954
topic           5266
dtype: int64
```

In [18]:
```
data.isna().sum()
```

Out[18]:
```
problem_text    5
topic           0
dtype: int64
```

In [19]:
```
data.groupby('problem_text').topic.nunique().sort_values(ascending=False).head()
```

Out[19]:
```
problem_text
It's okay. It's okay, it's okay, it's okay.
7
Each of the 102 pupils in one school is familiar with at least 68 others; prove that there are four of them who ha
ve the same number of acquaintances.                                                                          4
( a) Could it happen that in a company of 10 girls and 9 boys, all girls know different numbers of boys and all bo
ys know the same number of girls? (b) What if 11 girls and 10 boys?                                           4
How can n > 1 happen in a company of n + 1 girls and n boys all girls know different numbers of boys and all boys
know the same number of girls?                                                                                4
On the big chess board, 2n cells were marked so that the rooks can walk all the marked cells without jumping throu
gh unmarked cells. Prove that the figure from the observed cells can be cut into n rectangles.    4
Name: topic, dtype: int64
```

Видим:

- в датасете 29 полных дубликатов;
- при этом также имеется более 900 совпадающих текстов, имеющих при этом разные лэйблы.

Удалим полные дубликаты

In [20]:
```
data = data.drop_duplicates().reset_index(drop=True)
data.shape
```

Out[20]:
```
(5244, 2)
```

Так как наш датасет несбалансирован по классам, отранжируем датасет таким образом, чтобы наибольший ранг имел топик с наибольшим количеством текстов:

In [21]:
```
data.topic.value_counts().sort_values()
```

Out[21]:
```
topic
invariant         235
geometry          368
graphs            383
polynoms          417
dirichlet         441
combinatorics    1009
number_theory    2391
Name: count, dtype: int64
```

In [22]:
```
rang_dict = {k:v for v,k in enumerate(data.topic.value_counts().sort_values().index)}
data['rang_label'] = data.topic.map(rang_dict)
```

После сортировки по наиболее популярному рангу удалим дубликаты, сохранив каждое первое появление дубликата. Таким образом сохраним тексты для наименее популярных классов

In [23]:
```
data = data.sort_values(['problem_text','rang_label']
                        ).drop_duplicates('problem_text',keep='first'
                        ).reset_index(drop=True).drop(columns='rang_label')
data.shape
```

Out[23]:
```
(4319, 2)
```

In [24]:
```
data.topic.value_counts().sort_values()
```

Out[24]:
```
topic
invariant         235
geometry          367
graphs            379
dirichlet         390
polynoms          407
combinatorics     577
number_theory    1964
Name: count, dtype: int64
```

In [25]:
```
 data = data.dropna().reset_index(drop=True)
```

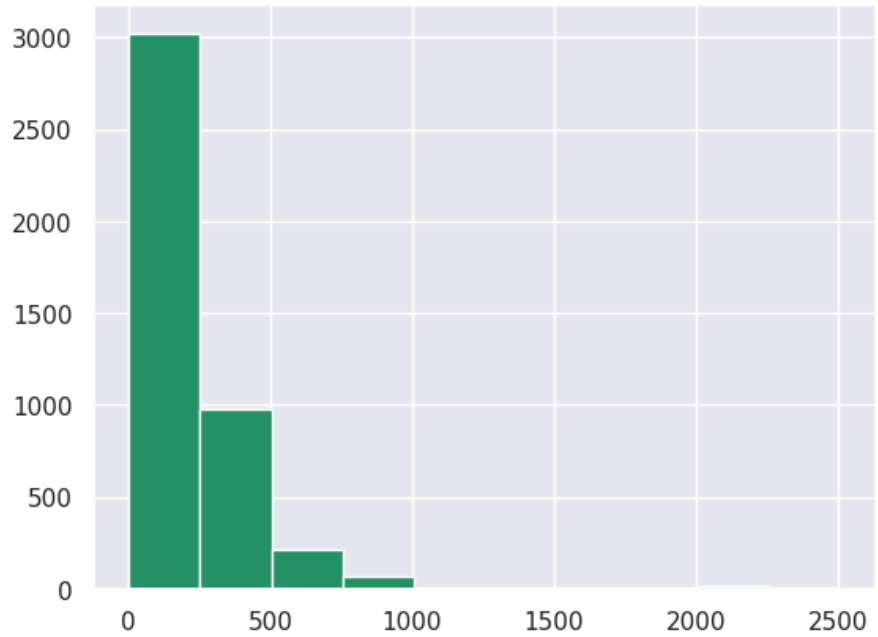
In [26]:
```
data.isna().sum()
```

Out[26]:
```
problem_text    0
topic           0
dtype: int64
```

Избавились от пропусков и дубликатов. Теперь посмотрим на статистики длины предложений в текстах

In [27]:
```
data['length_text'] = data.problem_text.apply(lambda x: len(str(x)))
data.length_text.hist()
```

Out[27]:
```
<Axes: >
```



In [28]:
```
data.describe()
```

Out[28]:

	length_text
count	4318.000000
mean	232.072024
std	229.677213
min	3.000000
25%	102.000000
50%	175.000000
75%	282.750000
max	2506.000000

In [29]:
```
#data = data.drop(columns='length_text')
#data.head()
```

Посмотрим на распределение данных поближе. Выведем основные квантили.

In [30]:
```
from scipy.stats import iqr

print(data.length_text.quantile(0.25)-iqr(data.length_text),
      data.length_text.quantile(0.25),
      data.length_text.quantile(0.5),
      data.length_text.quantile(0.75),
      data.length_text.quantile(0.75)+iqr(data.length_text))
```

```
-78.75 102.0 175.0 282.75 463.5
```

Видим, что:

- тексты > 500 символов являются аномальными для данного датасета. При этом короткие тексты составляют больше половины датасета;
- минимальная длина текста - 3 символа.

С учетом того, что общая тема датасета математическая, 3 символа может быть и достаточно для классификации. А, вот, если оставим аномальные длинные тексты, придется паддить довольные длинные последовательности, что неизбежно скажется на качестве модели.

Решил установить (ниже) ограничение по макс длине, а мин длину не ограничивать.

In [31]:
```
pip install -U pymorphy2-dicts-uk
```

```
Collecting pymorphy2-dicts-uk
  Downloading pymorphy2_dicts_uk-2.4.1.1.1460299261-py2.py3-none-any.whl.metadata (2.1 kB)
Downloading pymorphy2_dicts_uk-2.4.1.1.1460299261-py2.py3-none-any.whl (5.0 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.0/5.0 MB 57.2 MB/s eta 0:00:0000:0100:01
Installing collected packages: pymorphy2-dicts-uk
Successfully installed pymorphy2-dicts-uk-2.4.1.1.1460299261
Note: you may need to restart the kernel to use updated packages.
```

In [32]:
```
import pymorphy2
from pymorphy2 import MorphAnalyzer
```

Напишем функцию для предобработки текста

- удалим последовательности, состоящие из одних цифр;
- приведем текст к нижнему регистру;
- удалим знаки пунктуации и стоп-слова из текстов;
- токенизируем тексты;
- нормализуем лексемы;
- объединим получившися строки, добавив в начале каждой последовательности токенов начальный символ [CLS]

In [33]:
```
ENG_STOP_WORDS = set(stopwords.words('english')) # удалим стоп-слова
PUNCT_WORD_TOKENIZER = nltk.WordPunctTokenizer()
MORPH_ANALYZER = pymorphy2.MorphAnalyzer()

def preprocess_text(text):
    nums_filtered_text = re.sub(r'[0-9]+', '', text.lower())
    punct_filtered_text = ''.join(
        [ch for ch in nums_filtered_text if ch not in string.punctuation]
    )
    tokens = PUNCT_WORD_TOKENIZER.tokenize(punct_filtered_text)
    no_stopwords_filtered_tokens = [MORPH_ANALYZER.parse(token)[0].normal_form for token in tokens
                                    if token not in ENG_STOP_WORDS]
    norm_tokens = [MORPH_ANALYZER.parse(token)[0].normal_form for token in no_stopwords_filtered_tokens]

    return f"[CLS] {' '.join(norm_tokens)}"
```

Разобъем данные на train / test в соотношении 85%/15%, все перемешаем, стратифицируем выборку по лейблам

In [34]:
```
from sklearn.model_selection import train_test_split

label_to_ind = {topic:ind for ind,topic in enumerate(data.topic.unique())}
int_to_label = {ind: topic for topic, ind in label_to_ind.items()}
NUM_CLASSES = len(label_to_ind)

data['topic'] = data['topic'].map(label_to_ind)

train_data, eval_data = train_test_split(data[['problem_text','topic']], stratify=data['topic'], test_size=0.15, ra
```

Type *Markdown* and LaTeX: α^2

In [35]:
```
!nvidia-smi
DEVICE = 'cuda' if torch.cuda.is_available() else 'cpu'
DEVICE
```

```
Mon Dec  2 13:36:13 2024
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 560.35.03              Driver Version: 560.35.03      CUDA Version: 12.6     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  Tesla P100-PCIE-16GB           Off |   00000000:00:04.0 Off |                    0 |
| N/A   35C    P0             24W /  250W |       3MiB /  16384MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI        PID   Type   Process name                              GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
|  No running processes found                                                             |
+-----------------------------------------------------------------------------------------+
```

Out[35]:
```
'cuda'
```

Создадим кастомизированный класс MyDataset. В дальнейшем будем использовать токенайзер из репозитория huggingface. Макисмальная длина предложений - 500 символов; тексты, сод-е больше символов, будут обрезаны. Метод **getitem**(self, idx) возвращает: text_ids, attention_mask, torch.tensor(target).

In [36]:
```
MAX_LENGTH = 500

class MyDataset(Dataset):
  def __init__(self,
               my_dataset,
               tokenizer,
               device=DEVICE):

    self.text = my_dataset['problem_text'].apply(lambda x: preprocess_text(x)).tolist()
    self.target = my_dataset['topic'].tolist()
    self.tokenizer = tokenizer
    self.device = device

  def __getitem__(self, idx):
    prep_text = self.text[idx]
    target = self.target[idx]

    tokenized_text = self.tokenizer(text=prep_text,
                                    padding="max_length",
                                    max_length=MAX_LENGTH,
                                    truncation=True,
                                    return_tensors='pt')

    text_ids = tokenized_text['input_ids'].flatten()
    attention_mask = tokenized_text['attention_mask'].flatten()

    return text_ids, attention_mask, torch.tensor(target, dtype=torch.long)

  def __len__(self):
    return len(self.text)
```

In [53]:
```
!pip install GPUtil numba
from GPUtil import showUtilization as gpu_usage
import gc
```

```
huggingface/tokenizers: The current process just got forked, after parallelism has already been used. Disabling pa
rallelism to avoid deadlocks...
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)

Collecting GPUtil
  Downloading GPUtil-1.4.0.tar.gz (5.5 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: numba in /opt/conda/lib/python3.10/site-packages (0.60.0)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /opt/conda/lib/python3.10/site-packages (from numba)
(0.43.0)
Requirement already satisfied: numpy<2.1,>=1.22 in /opt/conda/lib/python3.10/site-packages (from numba) (1.26.4)
Building wheels for collected packages: GPUtil
  Building wheel for GPUtil (setup.py) ... done
  Created wheel for GPUtil: filename=GPUtil-1.4.0-py3-none-any.whl size=7394 sha256=3d7a26a840facda2589ef3343930d6
bad1b338f8173be75afed3f98f9e8089a2
  Stored in directory: /root/.cache/pip/wheels/a9/8a/bd/81082387151853ab8b6b3ef33426e98f5cbfebc3c397a9d4d0
Successfully built GPUtil
Installing collected packages: GPUtil
Successfully installed GPUtil-1.4.0
```

In [55]:
```
def free_gpu_cache():
    gc.collect()
    torch.cuda.empty_cache()
    print("GPU Usage after emptying the cache")
    gpu_usage()
free_gpu_cache()
```

```
GPU Usage after emptying the cache
| ID | GPU | MEM |
------------------
|  0 |  0% |  3% |
```

Задание 1 (2 балла)

Напишите кастомный класс для модели трансформера для задачи классификации, использующей в качествке backbone какую-то из моделей huggingface.

Т.е. конструктор класса должен принимать на вход название модели и подгружать её из huggingface, а затем использовать в качестве backbone (достаточно возможности использовать в качестве backbone те модели, которые упомянуты в последующих пунктах)

In [38]:
```
### This is just an interface example. You may change it if you want.

class TransformerClassificationModel(nn.Module):
    def __init__(self, base_transformer_model, num_classes: int=NUM_CLASSES):
        super().__init__()
        self.backbone = base_transformer_model
        # YOUR CODE: create additional layers for classfication
        hidden_size = self.backbone.config.hidden_size
        self.tanh = nn.Tanh()
        self.dropout = nn.Dropout(0.15)
        self.linear = nn.Linear(in_features=hidden_size, out_features=num_classes)

    def forward(self, input_ids, attention_mask):
        # YOUR CODE: propagate inputs through the model. Return dict with logits
        outputs = self.backbone(input_ids=input_ids, attention_mask=attention_mask)
        pooler_outputs = self.dropout(outputs.pooler_output)
        pooler_outputs = self.tanh(pooler_outputs)
        logits = self.linear(pooler_outputs)
        return logits, outputs.attentions
```

Задание 2 (1 балл)

Напишите функцию заморозки backbone у модели (если необходимо, возвращайте из функции модель)

In [39]:
```python
# заморозим веса последнего слоя
def freeze_backbone_function(model: TransformerClassificationModel):
    for param in model.backbone.parameters():
        param.requires_grad = False
    return model
```

Задание 3 (2 балла)

Напишите функцию, которая будет использована для тренировки (дообучения) трансформера (TransformerClassificationModel). Функция должна поддерживать обучение с замороженным и размороженным backbone.

In [40]:
```python
import evaluate
```

За основу возьмем код функции из семинарского ноутбука

In [41]:
```
acc_metric = evaluate.load("accuracy")

def train_transformer(transformer_model, n_epochs, train_dataloader, eval_dataloader, freeze_backbone=True, device=
    model = copy.deepcopy(transformer_model)
    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=1e-5)

    history_loss = {'train':[], 'valid':[]}
    history_acc = {'train':[], 'valid':[]}

    if freeze_backbone:
        model = freeze_backbone_function(model)

    for epoch_number in tqdm(range(n_epochs)):
        model.train()
        tr_loss = 0
        cnt_answers = 0
        cnt_right_answers = 0

        for data in train_dataloader:
            optimizer.zero_grad()

            input_data, attention_mask, labels = data
            input_data, attention_mask, labels = input_data.to(device), attention_mask.to(device), labels.to(device

            outputs,_ = model(input_data, attention_mask)

            pred = outputs.argmax(axis=-1)
            loss = criterion(outputs, labels)

            tr_loss += loss.item()
            cnt_answers += labels.shape[0]
            cnt_right_answers += (pred == labels).sum().item()

            loss.backward()
            optimizer.step()

        history_loss['train'].append(tr_loss / len(train_dataloader))
        history_acc['train'].append(cnt_right_answers / cnt_answers)

        model.eval()
        val_loss = 0
        vcnt_answers = 0
        vcnt_right_answers = 0

        with torch.no_grad():
            for vdata in tqdm(eval_dataloader):
                input_data, attention_mask, labels = vdata
                input_data, attention_mask, labels = input_data.to(device), attention_mask.to(device), labels.to(de

                outputs,_ = model(input_data, attention_mask)
                pred = outputs.argmax(axis=-1)
                vloss = criterion(outputs, labels)

                val_loss += vloss.item()
                vcnt_answers += labels.shape[0]
                vcnt_right_answers += (pred == labels).sum().item()

            history_loss['valid'].append(val_loss / len(eval_dataloader))
            history_acc['valid'].append(vcnt_right_answers / vcnt_answers)

            print('EPOCH {}:'.format(epoch_number + 1), "\n**TRAIN** loss:", tr_loss / len(train_dataloader),
                  "**TRAIN** Accuracy:", cnt_right_answers / cnt_answers,
                  "\n**EVAL** loss:", val_loss / len(eval_dataloader),
                  "**EVAL** Accuracy:", vcnt_right_answers / vcnt_answers,
                  '\n','-'*100
                 )
            del input_data, attention_mask, labels

        torch.cuda.empty_cache()
        gc.collect()

    return model, history_loss, history_acc
```

```
Downloading builder script:   0%|          | 0.00/4.20k [00:00<?, ?B/s]
```

Напишем функцию для отрисовки графиков прогресса обучения

In [42]:
```
def history_plot(history_loss, history_acc, name):
    fig, ax = plt.subplots(1,2, figsize=(15, 5))

    ax[0].plot(history_loss['train'], label="train_loss",color='lightblue')
    ax[0].plot(history_loss['valid'], label="val_loss",color='orange')
    ax[0].legend(['train', 'valid'])
    ax[0].set_xlabel('epochs')
    ax[0].set_ylabel("loss")
    plt.grid()

    ax[1].plot(history_acc['train'], label="train_acc",color='lightblue')
    ax[1].plot(history_acc['valid'], label="val_acc",color='orange')
    ax[1].legend(['train', 'valid'])
    ax[1].set_xlabel('epochs')
    ax[1].set_ylabel("accuracy")
    plt.grid()

    fig.suptitle(f'History of "{name}" model')
    plt.show()

MEAN_ACC_MODELS_HISTORY = {}
```

Задание 4 (1 балл)

Проверьте вашу функцию из предыдущего пункта, дообучив двумя способами *cointegrated/rubert-tiny2* из huggingface.

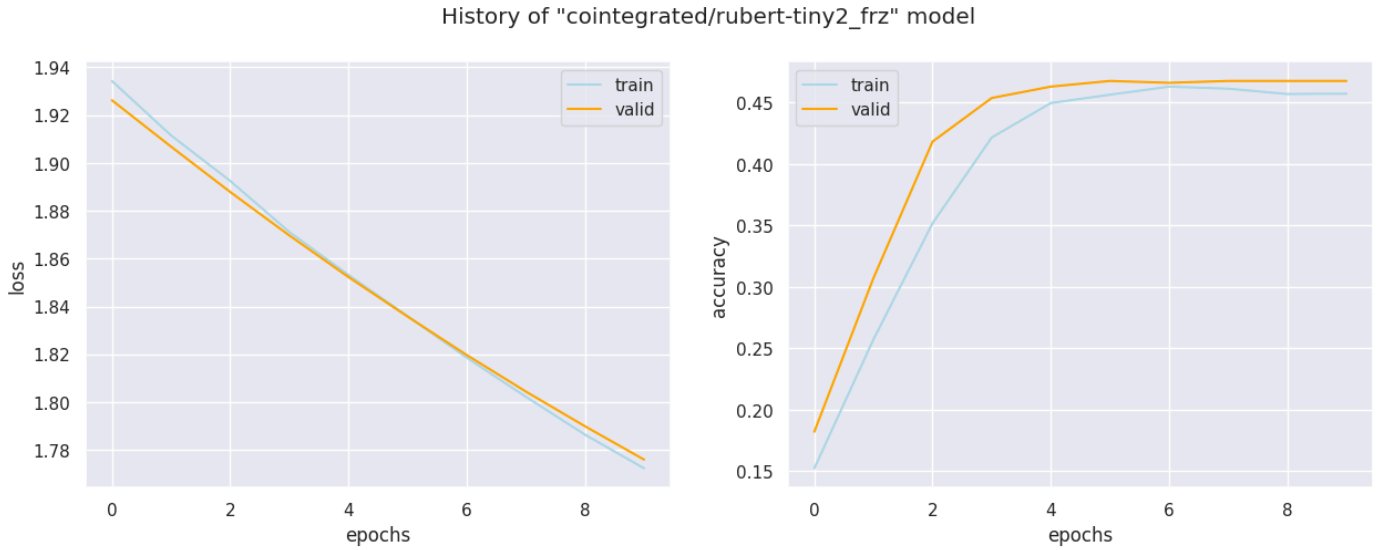
In [43]:
```
MODEL_NAME = 'cointegrated/rubert-tiny2'
MODEL_NAME_frz = MODEL_NAME +'_frz'
tokenizer_rt = transformers.AutoTokenizer.from_pretrained(MODEL_NAME)

train_dataset = MyDataset(train_data, tokenizer_rt)
eval_dataset = MyDataset(eval_data, tokenizer_rt)

train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
eval_loader = DataLoader(eval_dataset, batch_size=8, shuffle=True)
```

```
tokenizer_config.json:   0%|          | 0.00/401 [00:00<?, ?B/s]

vocab.txt:   0%|          | 0.00/1.08M [00:00<?, ?B/s]

tokenizer.json:   0%|          | 0.00/1.74M [00:00<?, ?B/s]

special_tokens_map.json:   0%|          | 0.00/112 [00:00<?, ?B/s]
```

In [46]:
```
EPOCHS = 10
pretrained_model_rt = BertModel.from_pretrained(MODEL_NAME, output_attentions=True)
model_rt = TransformerClassificationModel(pretrained_model_rt).to(DEVICE)
```

In [47]:
```
rubert_tiny_frz, rt_history_loss_frz, rt_history_acc_frz = train_transformer(transformer_model=model_rt
                                                                             ,n_epochs=EPOCHS
                                                                             ,train_dataloader=train_loader
                                                                             ,eval_dataloader=eval_loader
                                                                             ,freeze_backbone=True
                                                                             ,device=DEVICE)

history_plot(rt_history_loss_frz, rt_history_acc_frz, MODEL_NAME_frz)
MEAN_ACC_MODELS_HISTORY['accuracy'+MODEL_NAME_frz] = np.mean(rt_history_acc_frz['valid'])
```

```
  0%|          | 0/10 [00:00<?, ?it/s]

BertSdpaSelfAttention is used but `torch.nn.functional.scaled_dot_product_attention` does not support non-absolute
`position_embedding_type` or `output_attentions=True` or `head_mask`. Falling back to the manual attention impleme
ntation, but specifying the manual implementation will be required from Transformers version v5.0.0 onwards. This
warning can be removed using the argument `attn_implementation="eager"` when loading the model.

  0%|          | 0/81 [00:00<?, ?it/s]

EPOCH 1:
**TRAIN** loss: 1.9342926885770715 **TRAIN** Accuracy: 0.15231607629427793
**EVAL** loss: 1.9262260021986786 **EVAL** Accuracy: 0.18209876543209877
 ----------------------------------------------------------------------------------------------------

  0%|          | 0/81 [00:00<?, ?it/s]

EPOCH 2:
**TRAIN** loss: 1.9115293502807618 **TRAIN** Accuracy: 0.2572207084468665
**EVAL** loss: 1.9068301533475334 **EVAL** Accuracy: 0.30709876543209874
 ----------------------------------------------------------------------------------------------------

  0%|          | 0/81 [00:00<?, ?it/s]

EPOCH 3:
**TRAIN** loss: 1.8925172567367554 **TRAIN** Accuracy: 0.3517711171662125
**EVAL** loss: 1.8878861651008512 **EVAL** Accuracy: 0.4182098765432099
 ----------------------------------------------------------------------------------------------------

  0%|          | 0/81 [00:00<?, ?it/s]

EPOCH 4:
**TRAIN** loss: 1.8711672409721043 **TRAIN** Accuracy: 0.4215258855585831
**EVAL** loss: 1.8696956502066717 **EVAL** Accuracy: 0.4537037037037037
 ----------------------------------------------------------------------------------------------------

  0%|          | 0/81 [00:00<?, ?it/s]

EPOCH 5:
**TRAIN** loss: 1.853217583635579 **TRAIN** Accuracy: 0.44959128065395093
**EVAL** loss: 1.8523108193903794 **EVAL** Accuracy: 0.46296296296296297
 ----------------------------------------------------------------------------------------------------

  0%|          | 0/81 [00:00<?, ?it/s]

EPOCH 6:
**TRAIN** loss: 1.83594933644585 **TRAIN** Accuracy: 0.4564032697547684
**EVAL** loss: 1.8357904501903204 **EVAL** Accuracy: 0.4675925925925926
 ----------------------------------------------------------------------------------------------------

  0%|          | 0/81 [00:00<?, ?it/s]

EPOCH 7:
**TRAIN** loss: 1.8186022499333256 **TRAIN** Accuracy: 0.46294277929155314
**EVAL** loss: 1.8197270881982497 **EVAL** Accuracy: 0.4660493827160494
 ----------------------------------------------------------------------------------------------------

  0%|          | 0/81 [00:00<?, ?it/s]

EPOCH 8:
**TRAIN** loss: 1.8022594457087309 **TRAIN** Accuracy: 0.46130790190735693
**EVAL** loss: 1.8044445293921012 **EVAL** Accuracy: 0.4675925925925926
 ----------------------------------------------------------------------------------------------------

  0%|          | 0/81 [00:00<?, ?it/s]

EPOCH 9:
**TRAIN** loss: 1.786416799089183 **TRAIN** Accuracy: 0.45694822888283376
**EVAL** loss: 1.7899301361154627 **EVAL** Accuracy: 0.4675925925925926
 ----------------------------------------------------------------------------------------------------

  0%|          | 0/81 [00:00<?, ?it/s]

EPOCH 10:
**TRAIN** loss: 1.772337023589922 **TRAIN** Accuracy: 0.45722070844686646
**EVAL** loss: 1.775975446642181 **EVAL** Accuracy: 0.4675925925925926
 ----------------------------------------------------------------------------------------------------
```

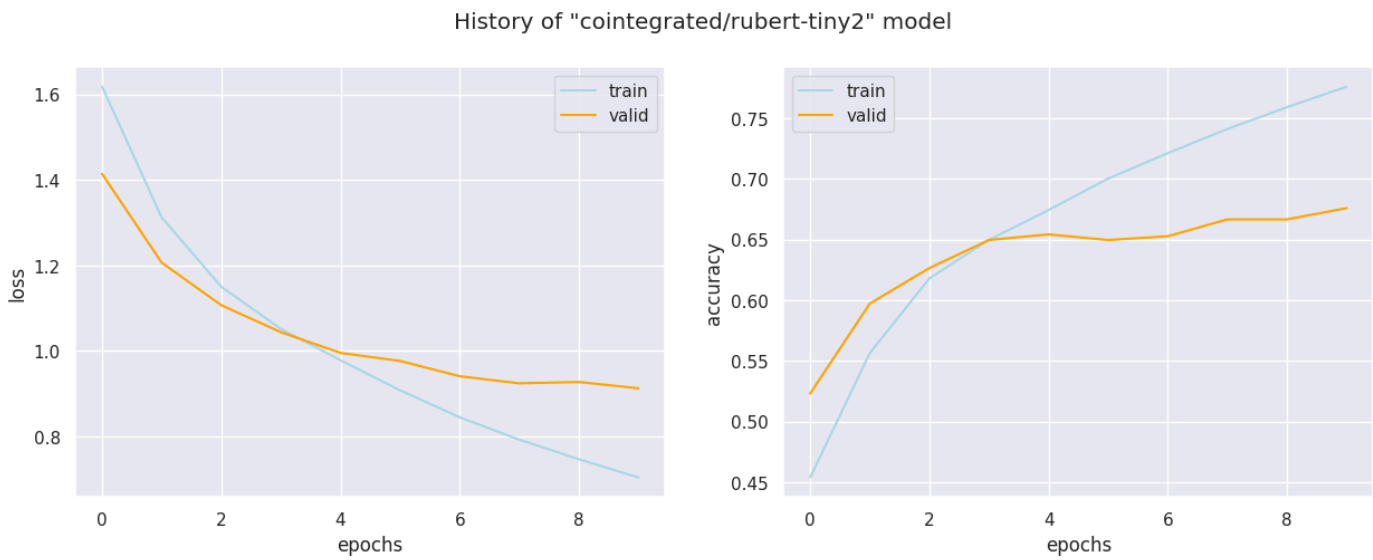


In [56]:
```
free_gpu_cache()
```

```
GPU Usage after emptying the cache
| ID | GPU | MEM |
------------------
|  0 |  0% |  3% |
```

In [57]:
```
rubert_tiny_unfrz, rt_history_loss_unfrz, rt_history_acc_unfrz = train_transformer(transformer_model=model_rt
                                                                                   ,n_epochs=EPOCHS
                                                                                   ,train_dataloader=train_loader
                                                                                   ,eval_dataloader=eval_loader
                                                                                   ,freeze_backbone=False
                                                                                   ,device=DEVICE)
history_plot(rt_history_loss_unfrz, rt_history_acc_unfrz, MODEL_NAME)
MEAN_ACC_MODELS_HISTORY['accuracy'+MODEL_NAME] = np.mean(rt_history_acc_unfrz['valid'])
```

```
  0%|          | 0/10 [00:00<?, ?it/s]

  0%|          | 0/81 [00:00<?, ?it/s]

EPOCH 1:
**TRAIN** loss: 1.6176244979319365 **TRAIN** Accuracy: 0.4539509536784741
**EVAL** loss: 1.4137803787066612 **EVAL** Accuracy: 0.5231481481481481
 ----------------------------------------------------------------------------------------------------

  0%|          | 0/81 [00:00<?, ?it/s]

EPOCH 2:
**TRAIN** loss: 1.3119974074156389 **TRAIN** Accuracy: 0.5564032697547684
**EVAL** loss: 1.2064341424423972 **EVAL** Accuracy: 0.5972222222222222
 ----------------------------------------------------------------------------------------------------

  0%|          | 0/81 [00:00<?, ?it/s]

EPOCH 3:
**TRAIN** loss: 1.1501189612823983 **TRAIN** Accuracy: 0.6179836512261581
**EVAL** loss: 1.1069969277323028 **EVAL** Accuracy: 0.6265432098765432
 ----------------------------------------------------------------------------------------------------

  0%|          | 0/81 [00:00<?, ?it/s]

EPOCH 4:
**TRAIN** loss: 1.0515945980082388 **TRAIN** Accuracy: 0.6498637602179836
**EVAL** loss: 1.0439013603292866 **EVAL** Accuracy: 0.6496913580246914
 ----------------------------------------------------------------------------------------------------

  0%|          | 0/81 [00:00<?, ?it/s]

EPOCH 5:
**TRAIN** loss: 0.9783068244871886 **TRAIN** Accuracy: 0.6743869209809265
**EVAL** loss: 0.9955341455377178 **EVAL** Accuracy: 0.654320987654321
 ----------------------------------------------------------------------------------------------------

  0%|          | 0/81 [00:00<?, ?it/s]

EPOCH 6:
**TRAIN** loss: 0.908111641458843 **TRAIN** Accuracy: 0.7002724795640327
**EVAL** loss: 0.9766946681487707 **EVAL** Accuracy: 0.6496913580246914
 ----------------------------------------------------------------------------------------------------

  0%|          | 0/81 [00:00<?, ?it/s]

EPOCH 7:
**TRAIN** loss: 0.845131196146426 **TRAIN** Accuracy: 0.7212534059945505
**EVAL** loss: 0.9409612851378358 **EVAL** Accuracy: 0.6527777777777778
 ----------------------------------------------------------------------------------------------------

  0%|          | 0/81 [00:00<?, ?it/s]

EPOCH 8:
**TRAIN** loss: 0.7927544084580048 **TRAIN** Accuracy: 0.7411444141689373
**EVAL** loss: 0.9244897452033596 **EVAL** Accuracy: 0.6666666666666666
 ----------------------------------------------------------------------------------------------------

  0%|          | 0/81 [00:00<?, ?it/s]

EPOCH 9:
**TRAIN** loss: 0.7471047896405925 **TRAIN** Accuracy: 0.7591280653950954
**EVAL** loss: 0.9275640074485614 **EVAL** Accuracy: 0.6666666666666666
 ----------------------------------------------------------------------------------------------------

  0%|          | 0/81 [00:00<?, ?it/s]

EPOCH 10:
**TRAIN** loss: 0.7043489725693413 **TRAIN** Accuracy: 0.7760217983651226
**EVAL** loss: 0.912750001122922 **EVAL** Accuracy: 0.6759259259259259
 ----------------------------------------------------------------------------------------------------
```



In [58]:
```
free_gpu_cache()
```

```
GPU Usage after emptying the cache
| ID | GPU | MEM |
------------------
|  0 | 19% |  5% |
```

In [59]:
```
MEAN_ACC_MODELS_HISTORY
```

Out[59]:
```
{'accuracycointegrated/rubert-tiny2_frz': 0.41604938271604935,
 'accuracycointegrated/rubert-tiny2': 0.6362654320987654}
```

Задание 5 (1 балл)

Обучите *tbs17/MathBert* (с замороженным backbone и без заморозки), проанализируйте результаты. Сравните скоры с первым заданием. Получилось лучше или нет? Почему?

In [60]:
```
class MathBertClassificationModel(nn.Module):
    def __init__(self, base_transformer_model, num_classes: int=NUM_CLASSES):
        super().__init__()
        self.backbone = base_transformer_model
        self.linear = nn.Linear(in_features=self.backbone.config.hidden_size, out_features=num_classes)


    def forward(self, input_ids, attention_mask):
        # YOUR CODE: propagate inputs through the model. Return dict with logits
        outputs = self.backbone(input_ids=input_ids, attention_mask=attention_mask)
        logits = self.linear(outputs.pooler_output)

        return logits, outputs.attentions
```

In [61]:
```
MODEL_NAME = 'tbs17/MathBert'
MODEL_NAME_frz = MODEL_NAME +'_frz'
tokenizer_mb = transformers.AutoTokenizer.from_pretrained(MODEL_NAME)

train_dataset = MyDataset(train_data, tokenizer_mb)
eval_dataset = MyDataset(eval_data, tokenizer_mb)

train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
eval_loader = DataLoader(eval_dataset, batch_size=8, shuffle=True)
```

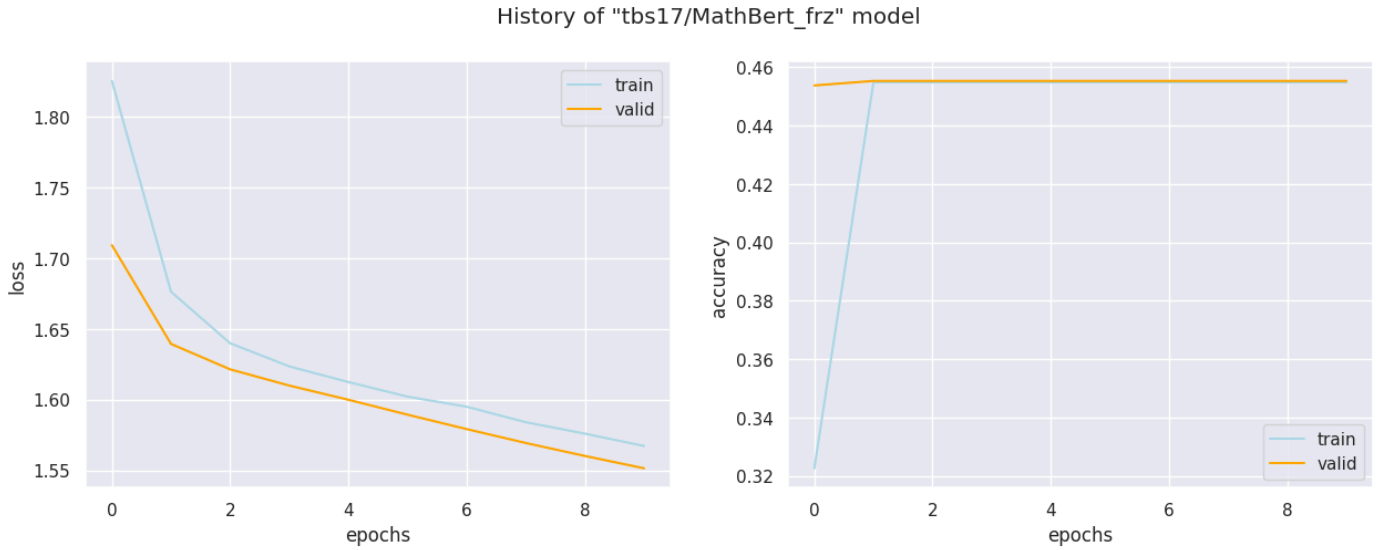
```
tokenizer_config.json:   0%|          | 0.00/28.0 [00:00<?, ?B/s]

config.json:   0%|          | 0.00/569 [00:00<?, ?B/s]

vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]

tokenizer.json:   0%|          | 0.00/466k [00:00<?, ?B/s]
```

In [62]:
```
pretrained_model_mb = BertModel.from_pretrained(MODEL_NAME, output_attentions=True)
model_mb = MathBertClassificationModel(pretrained_model_mb).to(DEVICE)
```

```
pytorch_model.bin:   0%|          | 0.00/441M [00:00<?, ?B/s]
```

In [63]:
```
mathbert_frz, mb_history_loss_frz, mb_history_acc_frz = train_transformer(transformer_model=model_mb
                                                                          ,n_epochs=EPOCHS
                                                                          ,train_dataloader=train_loader
                                                                          ,eval_dataloader=eval_loader
                                                                          ,freeze_backbone=True
                                                                          ,device=DEVICE)

history_plot(mb_history_loss_frz, mb_history_acc_frz, MODEL_NAME_frz)
MEAN_ACC_MODELS_HISTORY['accuracy'+MODEL_NAME_frz] = np.mean(mb_history_acc_frz['valid'])
```

```
  0%|          | 0/10 [00:00<?, ?it/s]

  0%|          | 0/81 [00:00<?, ?it/s]

EPOCH 1:
**TRAIN** loss: 1.8253678596538045 **TRAIN** Accuracy: 0.3226158038147139
**EVAL** loss: 1.7091996919961623 **EVAL** Accuracy: 0.4537037037037037
 ----------------------------------------------------------------------------------------------------

  0%|          | 0/81 [00:00<?, ?it/s]

EPOCH 2:
**TRAIN** loss: 1.676224374771118 **TRAIN** Accuracy: 0.4547683923705722
**EVAL** loss: 1.639345692999569 **EVAL** Accuracy: 0.45524691358024694
 ----------------------------------------------------------------------------------------------------

  0%|          | 0/81 [00:00<?, ?it/s]

EPOCH 3:
**TRAIN** loss: 1.639827353021373 **TRAIN** Accuracy: 0.4547683923705722
**EVAL** loss: 1.6213712589240368 **EVAL** Accuracy: 0.45524691358024694
 ----------------------------------------------------------------------------------------------------

  0%|          | 0/81 [00:00<?, ?it/s]

EPOCH 4:
**TRAIN** loss: 1.6234628558158875 **TRAIN** Accuracy: 0.4547683923705722
**EVAL** loss: 1.6098909436920543 **EVAL** Accuracy: 0.45524691358024694
 ----------------------------------------------------------------------------------------------------

  0%|          | 0/81 [00:00<?, ?it/s]

EPOCH 5:
**TRAIN** loss: 1.6123731229616247 **TRAIN** Accuracy: 0.4547683923705722
**EVAL** loss: 1.5998532028845798 **EVAL** Accuracy: 0.45524691358024694
 ----------------------------------------------------------------------------------------------------

  0%|          | 0/81 [00:00<?, ?it/s]

EPOCH 6:
**TRAIN** loss: 1.6020637398180755 **TRAIN** Accuracy: 0.4547683923705722
**EVAL** loss: 1.5893026450533925 **EVAL** Accuracy: 0.45524691358024694
 ----------------------------------------------------------------------------------------------------

  0%|          | 0/81 [00:00<?, ?it/s]

EPOCH 7:
**TRAIN** loss: 1.5949428858964338 **TRAIN** Accuracy: 0.4547683923705722
**EVAL** loss: 1.5791067944632635 **EVAL** Accuracy: 0.45524691358024694
 ----------------------------------------------------------------------------------------------------

  0%|          | 0/81 [00:00<?, ?it/s]

EPOCH 8:
**TRAIN** loss: 1.5839632112046946 **TRAIN** Accuracy: 0.4547683923705722
**EVAL** loss: 1.56928159646046 **EVAL** Accuracy: 0.45524691358024694
 ----------------------------------------------------------------------------------------------------

  0%|          | 0/81 [00:00<?, ?it/s]

EPOCH 9:
**TRAIN** loss: 1.5758786636850108 **TRAIN** Accuracy: 0.4547683923705722
**EVAL** loss: 1.5601069000032213 **EVAL** Accuracy: 0.45524691358024694
 ----------------------------------------------------------------------------------------------------

  0%|          | 0/81 [00:00<?, ?it/s]

EPOCH 10:
**TRAIN** loss: 1.5671651599199876 **TRAIN** Accuracy: 0.4547683923705722
**EVAL** loss: 1.5513829309263347 **EVAL** Accuracy: 0.45524691358024694
 ----------------------------------------------------------------------------------------------------
```



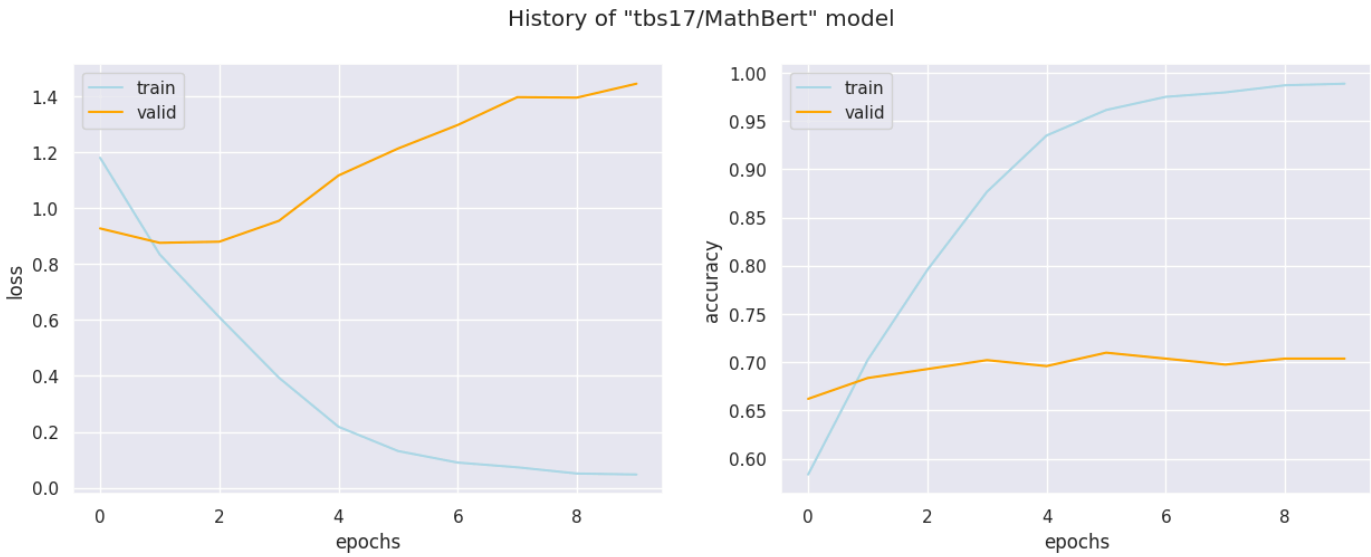
In [64]:
```
free_gpu_cache()
```

```
GPU Usage after emptying the cache
| ID | GPU | MEM |
------------------
|  0 |  0% | 10% |
```

In [65]:

```
mathbert_unfrz, mathbert_history_loss_unfrz, mathbert_history_acc_unfrz = train_transformer(transformer_model=model
                                                                                            ,n_epochs=EPOCHS
                                                                                            ,train_dataloader=train
                                                                                            ,eval_dataloader=eval_l
                                                                                            ,freeze_backbone=False
                                                                                            ,device=DEVICE)

history_plot(mathbert_history_loss_unfrz, mathbert_history_acc_unfrz, MODEL_NAME)
MEAN_ACC_MODELS_HISTORY['accuracy'+MODEL_NAME] = np.mean(mathbert_history_acc_unfrz['valid'])
```

```
  0%|          | 0/10 [00:00<?, ?it/s]

  0%|          | 0/81 [00:00<?, ?it/s]

EPOCH 1:
**TRAIN** loss: 1.1803851879161338 **TRAIN** Accuracy: 0.5836512261580381
**EVAL** loss: 0.9275918041850314 **EVAL** Accuracy: 0.6620370370370371
 ----------------------------------------------------------------------------------------------------

  0%|          | 0/81 [00:00<?, ?it/s]

EPOCH 2:
**TRAIN** loss: 0.8347394031027089 **TRAIN** Accuracy: 0.7024523160762943
**EVAL** loss: 0.8760351543257265 **EVAL** Accuracy: 0.683641975308642
 ----------------------------------------------------------------------------------------------------

  0%|          | 0/81 [00:00<?, ?it/s]

EPOCH 3:
**TRAIN** loss: 0.6102691039766954 **TRAIN** Accuracy: 0.7956403269754768
**EVAL** loss: 0.8801130650588024 **EVAL** Accuracy: 0.6929012345679012
 ----------------------------------------------------------------------------------------------------

  0%|          | 0/81 [00:00<?, ?it/s]

EPOCH 4:
**TRAIN** loss: 0.39376149587333203 **TRAIN** Accuracy: 0.876566757493188
**EVAL** loss: 0.9547646821097091 **EVAL** Accuracy: 0.7021604938271605
 ----------------------------------------------------------------------------------------------------

  0%|          | 0/81 [00:00<?, ?it/s]

EPOCH 5:
**TRAIN** loss: 0.21822033649229486 **TRAIN** Accuracy: 0.9348773841961853
**EVAL** loss: 1.1168705001013515 **EVAL** Accuracy: 0.6959876543209876
 ----------------------------------------------------------------------------------------------------

  0%|          | 0/81 [00:00<?, ?it/s]

EPOCH 6:
**TRAIN** loss: 0.13139072374400237 **TRAIN** Accuracy: 0.9613079019073569
**EVAL** loss: 1.2134498435727976 **EVAL** Accuracy: 0.7098765432098766
 ----------------------------------------------------------------------------------------------------

  0%|          | 0/81 [00:00<?, ?it/s]

EPOCH 7:
**TRAIN** loss: 0.09011604285434537 **TRAIN** Accuracy: 0.9749318801089918
**EVAL** loss: 1.2974518929366712 **EVAL** Accuracy: 0.7037037037037037
 ----------------------------------------------------------------------------------------------------

  0%|          | 0/81 [00:00<?, ?it/s]

EPOCH 8:
**TRAIN** loss: 0.07326970762451705 **TRAIN** Accuracy: 0.9795640326975477
**EVAL** loss: 1.397071680456492 **EVAL** Accuracy: 0.6975308641975309
 ----------------------------------------------------------------------------------------------------

  0%|          | 0/81 [00:00<?, ?it/s]

EPOCH 9:
**TRAIN** loss: 0.050942077316627234 **TRAIN** Accuracy: 0.9869209809264305
**EVAL** loss: 1.3954701243359366 **EVAL** Accuracy: 0.7037037037037037
 ----------------------------------------------------------------------------------------------------

  0%|          | 0/81 [00:00<?, ?it/s]

EPOCH 10:
**TRAIN** loss: 0.04743745292273714 **TRAIN** Accuracy: 0.9885558583106268
**EVAL** loss: 1.445205957053896 **EVAL** Accuracy: 0.7037037037037037
 ----------------------------------------------------------------------------------------------------
```



In [66]:
```
MEAN_ACC_MODELS_HISTORY
```

Out[66]:
```
{'accuracycointegrated/rubert-tiny2_frz': 0.41604938271604935,
 'accuracycointegrated/rubert-tiny2': 0.6362654320987654,
 'accuracytbs17/MathBert_frz': 0.45509259259259266,
 'accuracytbs17/MathBert': 0.6955246913580249}
```

Выводы по обучению моделей:

1. Для всех моделей справедливо, что дообучение улучшает их качество: лосс падает, точность прогноза растет.
2. **rubert-tiny2** с "расмороженным" добавленным линейным слоем обучается заметно быстрее по сравнению с **MathBert** с аналогичной архитектурой; при этом первая модель ожидаемо уступает второй, специализированной для нашей задачи модели.
3. При этом модели с "размороженными" слоями дообученной архитектуры имеют склонность к переобучению. Разница особенно заметна на графиках обучения **MathBert**: при стабильной accuracy на валидации loss постепенно увеличивается, при этом accuracy на трейне стремится к 100%.
Полагаю, это связано с тем, что в последнем эксперименте убрал hidenlayer + tanh функцию активации, добавив просто линейный слой с выходами для всех классов. До этого, с дропаутом - 15%, среднее качество составляло около 71 %.
4. Очевидно, что fine-tuning с "разморозкой" последнего слоя помогает сделать backbone model более подходящей к стоящей перед ней задаче. Однако выбранный размер батчей (8) для eval + недостаток времени для проведения экспериментов не позволил достичь оптимальной сходимости...Хотя **MathBert** с замороженным слоем сходится на 2-й эпохи к accuracy = 0.45 и перестает дальше обучаться)

Задание 6 (1 балл)

Напишите функцию для отрисовки карт внимания первого слоя для моделей из задания

In [251]:
```
from transformers import AutoTokenizer, AutoModel, utils
```

In [246]:
```
def plot_attention(text_loader, model, device=DEVICE):
    model.eval()
    model.backbone.config.output_attentions = True
    with torch.no_grad():
        text = next(iter(text_loader.text))
        tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")

        inputs = tokenizer.encode_plus(text, return_tensors='pt')
        input_ids = inputs['input_ids'].to(device)
        attention_mask = inputs['attention_mask'].to(device)
        outputs = model(input_ids, attention_mask=attention_mask)
        first_layer_attentions = outputs[-1]
        tokens = tokenizer.convert_ids_to_tokens(input_ids[0])
        return first_layer_attentions, tokens
```

Model View

Просмотр модели позволяет увидеть всю модель с "высоты птичьего полета". В каждой ячейке отображаются значения концентрации внимания для конкретной "головы", проиндексированные по слоям (строкам) и "головам" (столбцам). Линии в каждой ячейке отражают внимание от одного токена (слева) к другому (справа), при этом вес линий пропорционален значению.

In [247]:
```
brt_base = BertModel.from_pretrained("cointegrated/rubert-tiny2", output_attentions=True)
rubert_tiny_model = TransformerClassificationModel(brt_base, 7).to(DEVICE)
attention, tokens = plot_attention(text_loader=train_dataset, model=rubert_tiny_model)
model_view(attention, tokens)
```

```
<IPython.core.display.Javascript object>
```

Head View

The head view visualizes attention in one or more heads from a single Transformer layer. Each line shows the attention from one token (left) to another (right). Line weight reflects the attention value (ranges from 0 to 1), while line color identifies the attention head. When multiple heads are selected (indicated by the colored tiles at the top), the corresponding visualizations are overlaid onto one another. For a more detailed explanation of attention in Transformer models, please refer to the blog (https://towardsdatascience.com/deconstructing-bert-part-2-visualizing-the-inner-workings-of-attention-60a16d86b5c1).

Usage

- **Hover** over any **token** on the left/right side of the visualization to filter attention from/to that token.
- **Double-click** on any of the **colored tiles** at the top to filter to the corresponding attention head.
- **Single-click** on any of the **colored tiles** to toggle selection of the corresponding attention head.
- **Click** on the **Layer** drop-down to change the model layer (zero-indexed).

In [248]:
```
head_view(attention, tokens)
```

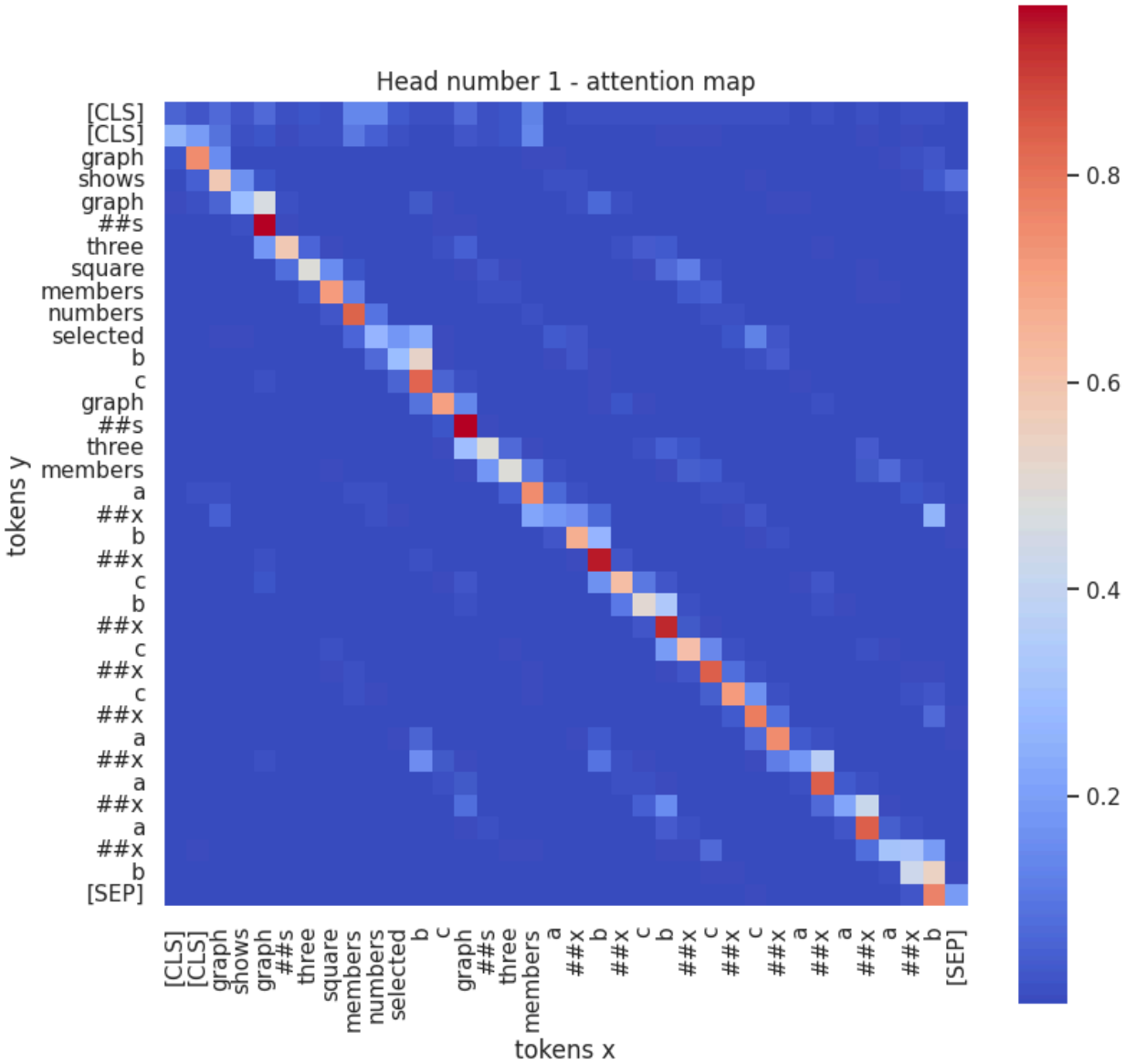
Layer: 

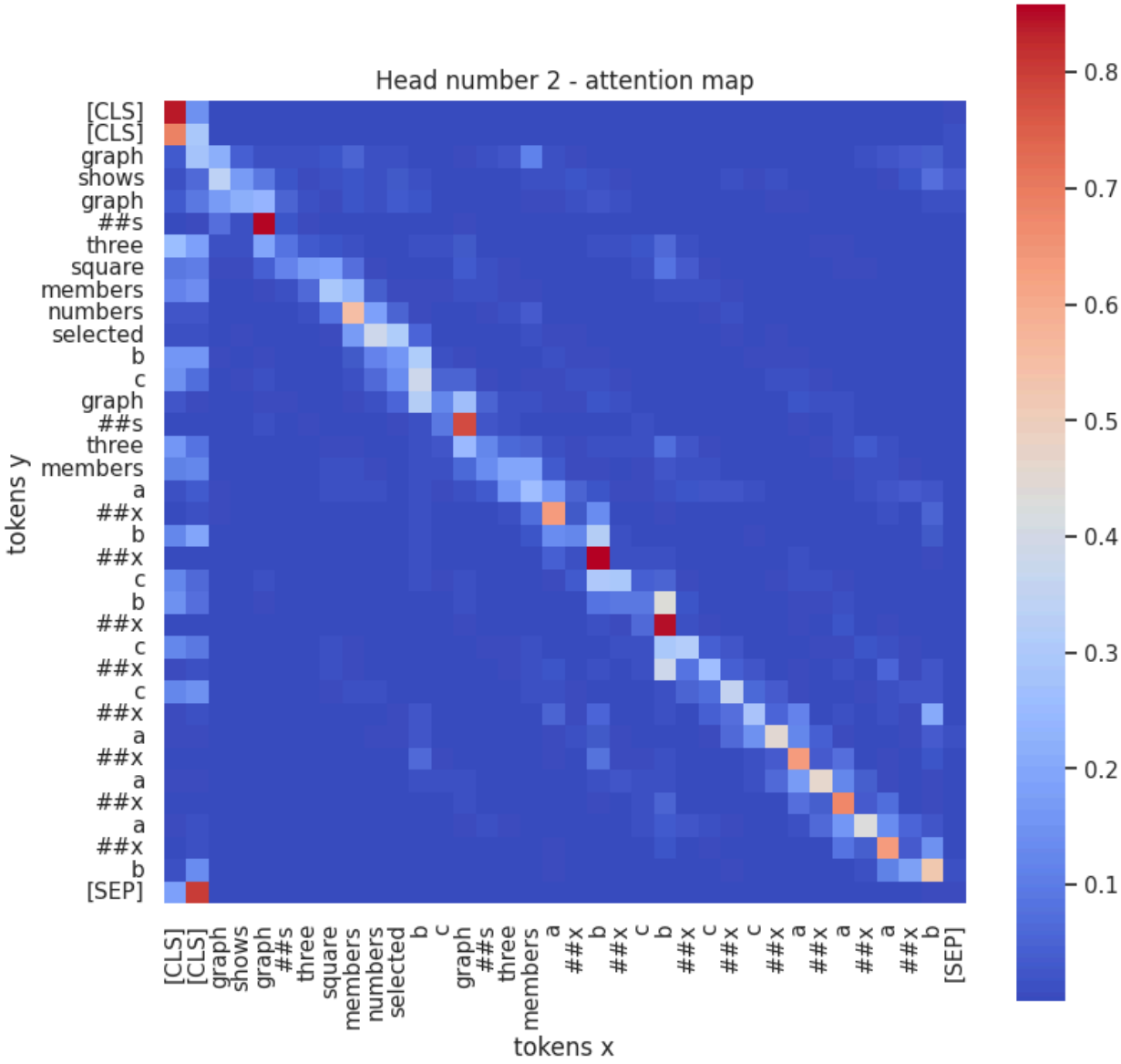
```
<IPython.core.display.Javascript object>
```

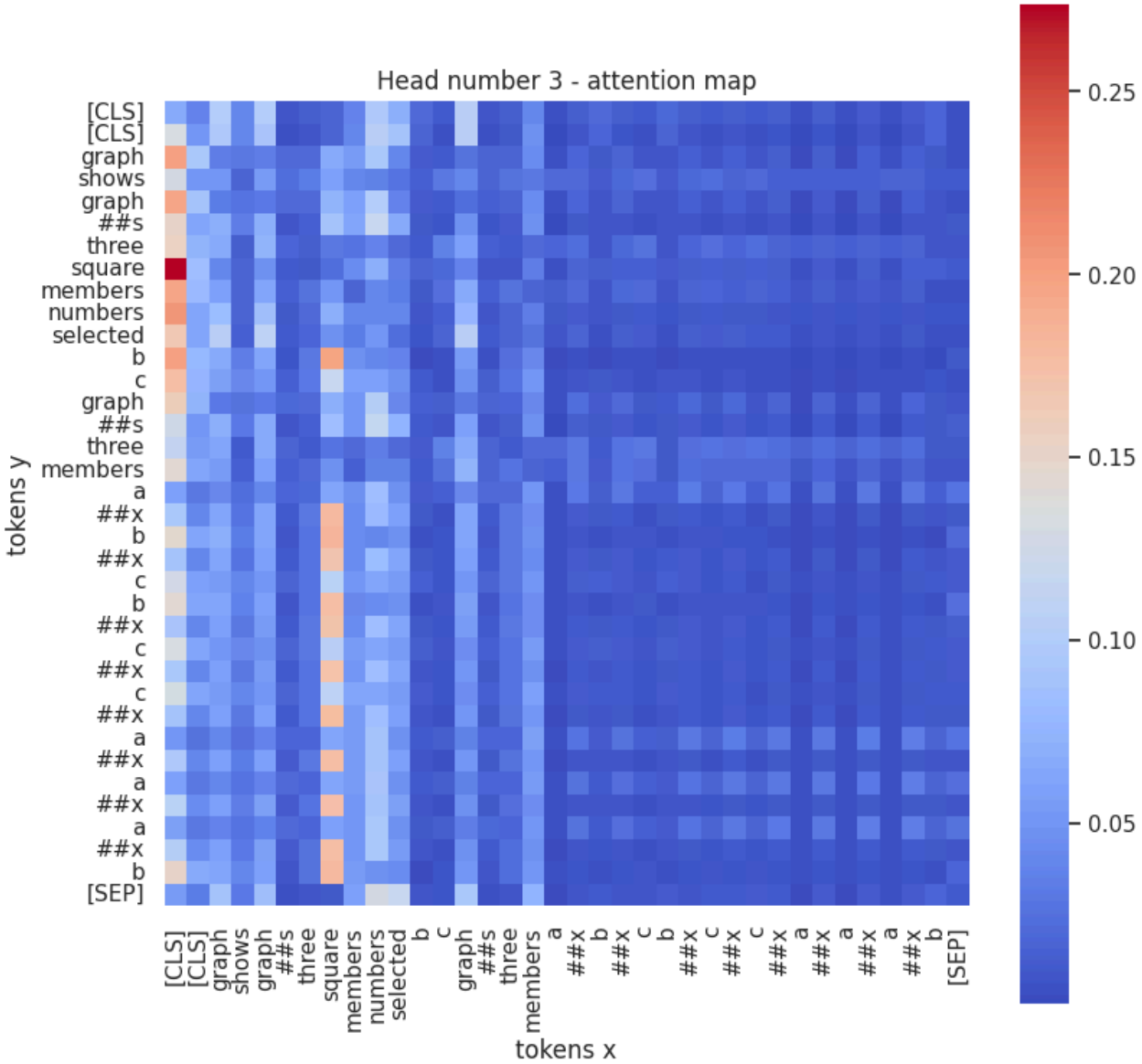
In [249]:
```
def draw_first_layer_attention_maps(text_loader, model, tokenizer, part_of_heads=0.4, device=DEVICE):
    model.eval()
    model.backbone.config.output_attentions = True
    with torch.no_grad():
        text = next(iter(text_loader.text))

        inputs = tokenizer(text, return_tensors='pt')
        input_data = inputs['input_ids'].to(device)
        attention_mask = inputs['attention_mask'].to(device)

        _, attentions = model(input_ids=input_data, attention_mask=attention_mask)

        first_layer_attentions = attentions[0]
        tokens = tokenizer.convert_ids_to_tokens(input_data[0])
        num_heads = first_layer_attentions.size(1)

        print(f"Голов Attention = {num_heads}, ниже отрисовка {part_of_heads*100}% из них")
        num_heads = int(num_heads*part_of_heads)
        for head_id in range(num_heads):

            attention = first_layer_attentions[0, head_id].cpu().detach().numpy()
            plt.figure(figsize=(9, 9))
            sns.heatmap(attention, annot=False, square=True, cmap='coolwarm',xticklabels=tokens, yticklabels=tokens
            plt.title(f"Head number {head_id+1} - attention map")
            plt.xlabel("tokens x")
            plt.ylabel("tokens y")
            plt.show()

        model.backbone.config.output_attentions = False
```

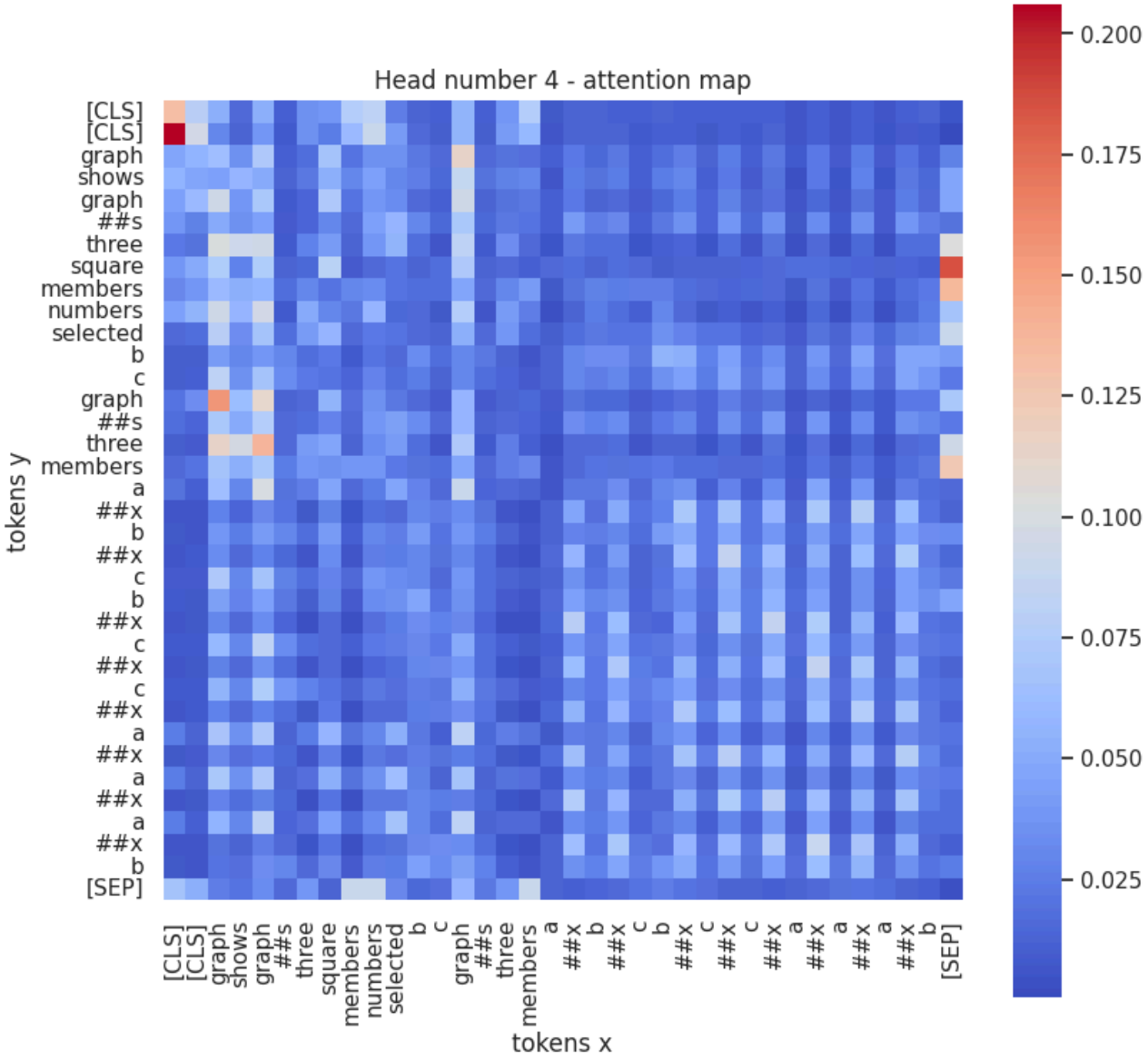
In [252]:
```
#brt_base = BertModel.from_pretrained("cointegrated/rubert-tiny2")
#rubert_tiny_model = TransformerClassificationModel(brt_base, 7).to(DEVICE)
draw_first_layer_attention_maps(text_loader=train_dataset, model=rubert_tiny_model, tokenizer=tokenizer_rt)
```

```
Голов Attention = 12, ниже отрисовка 40.0% из них
```




Head number 2 - attention map
tokens y
[CLS]
[CLS]
graph
shows
graph
##s
three
square
members
numbers
selected
b
c
graph
##s
three
members
a
##x
b
##x
c
b
##x
c
##x
c
##x
a
##x
a
##x
a
##x
b
[SEP]
tokens x
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1


Head number 3 - attention map
tokens y
[CLS]
[CLS]
graph
shows
graph
##s
three
square
members
numbers
selected
b
c
graph
##s
three
members
a
##x
b
##x
c
b
##x
c
##x
c
##x
a
##x
a
##x
a
##x
b
[SEP]
tokens x
0.25
0.20
0.15
0.10
0.05



In [69]:
```
next(iter(train_dataset.text))
```

Out[69]:
```
'[CLS] graph shows graphs three square members numbers selected b c graphs three members ax bx c bx cx cx ax ax ax b'
```

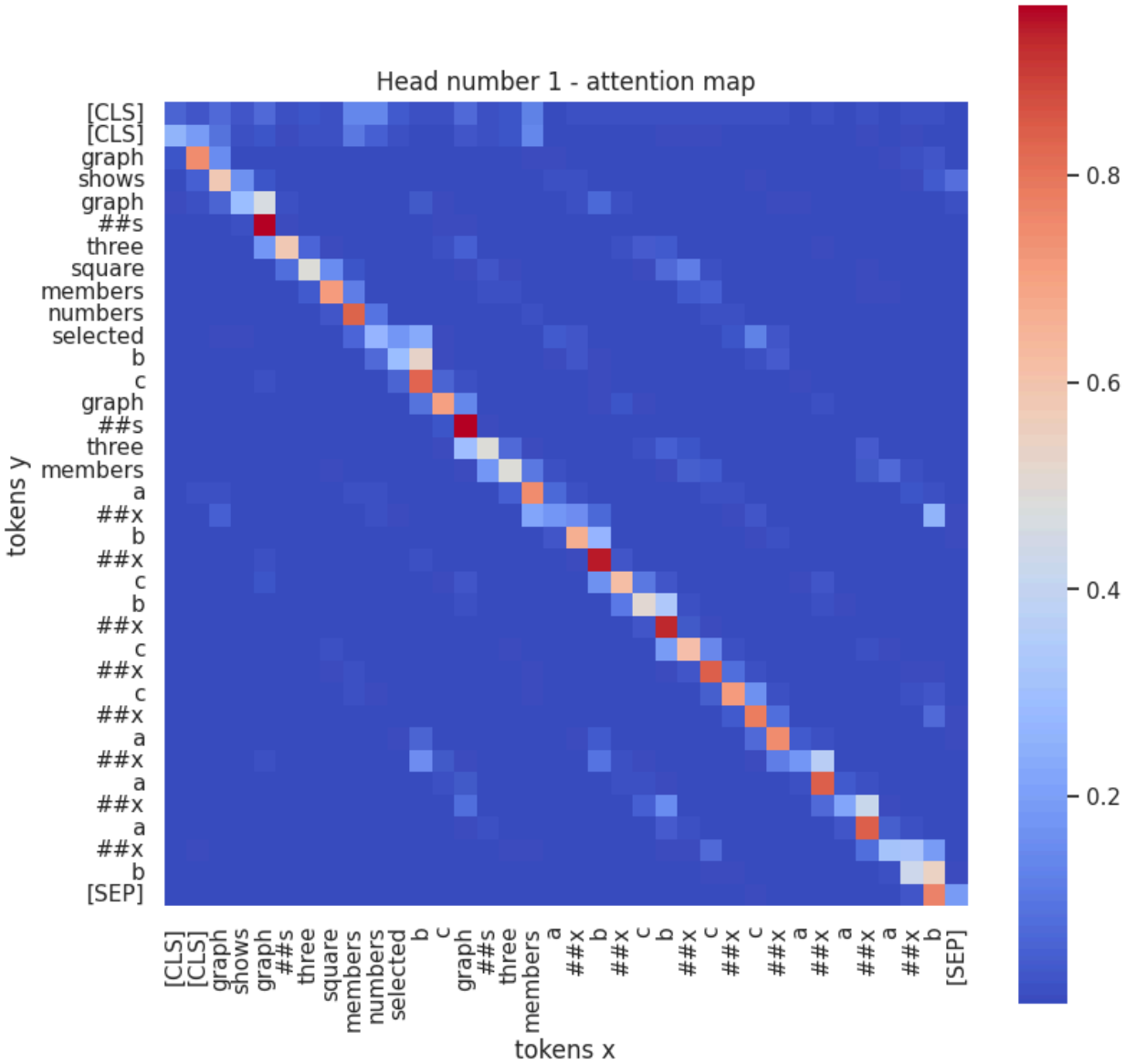
Задание 7 (1 балл)

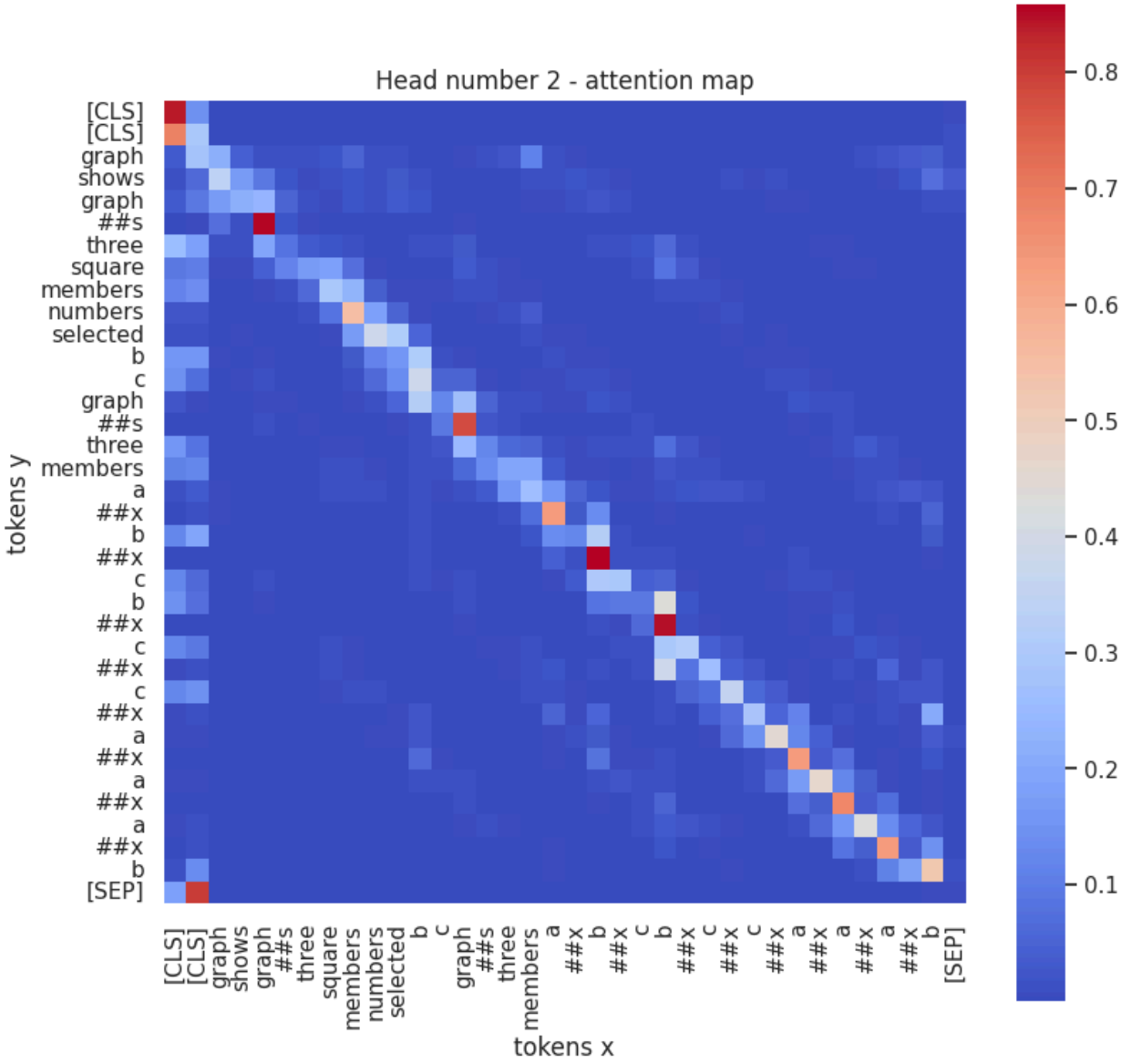
Проведите инференс для всех моделей **ДО ДООБУЧЕНИЯ** на 2-3 текстах из датасета. Посмотрите на головы Attention первого слоя в каждой модели на выбранных текстах (отрисуйте их отдельно).

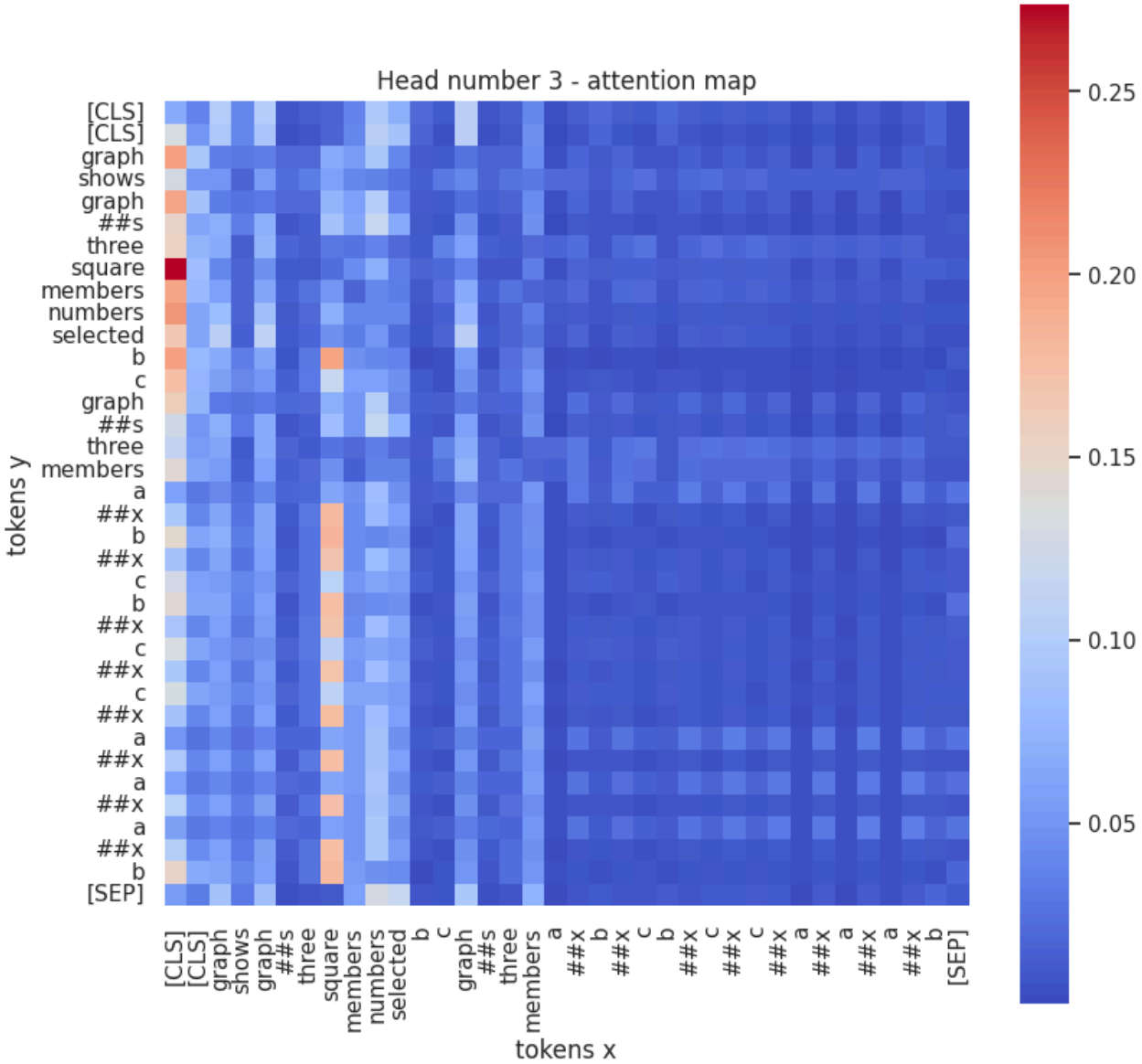
Попробуйте их проинтерпретировать. Какие связи улавливают карты внимания? (если в модели много голов Attention, то проинтерпретируйте наиболее интересные)

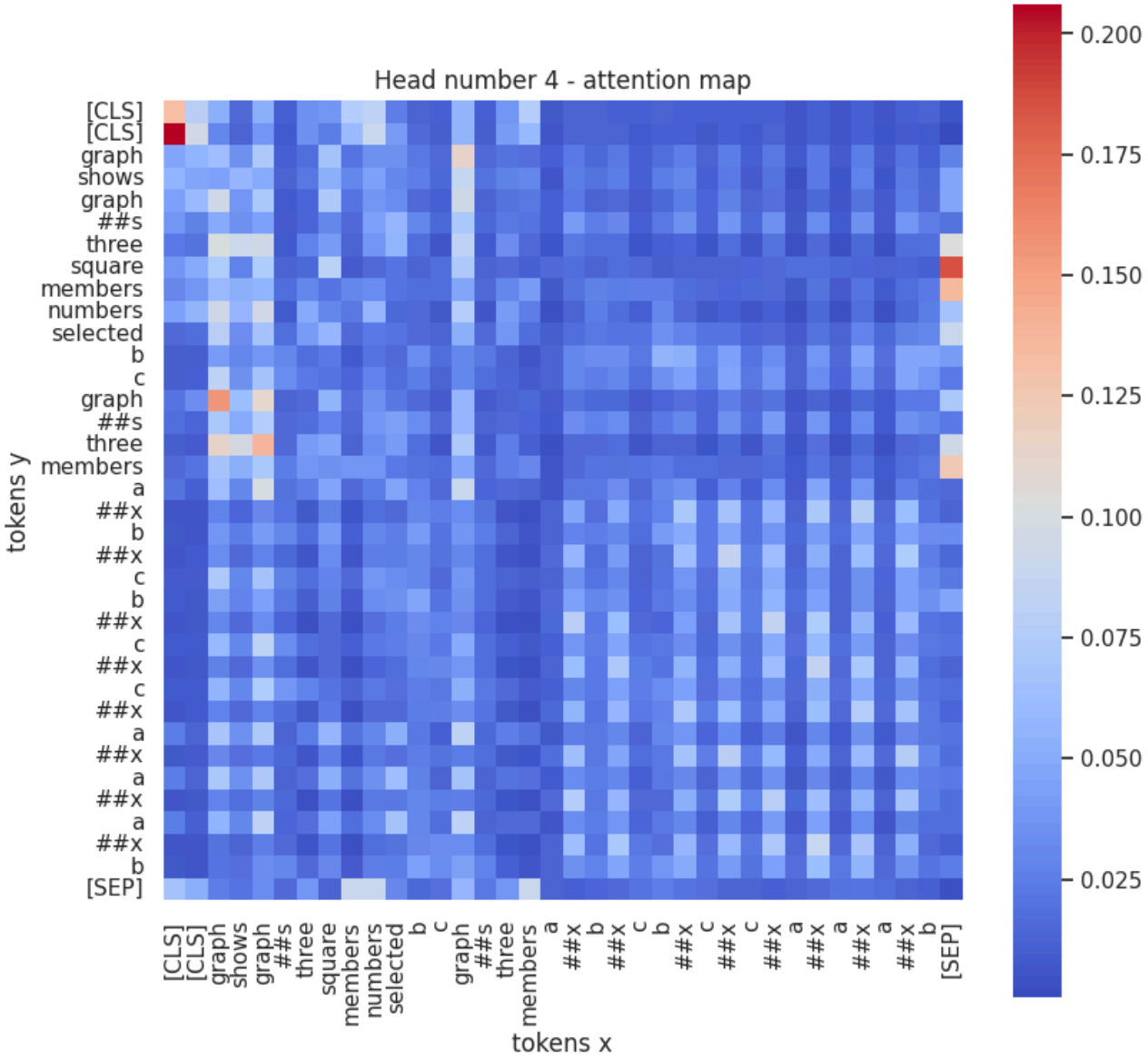
In [253]:
```
brt_base = BertModel.from_pretrained("cointegrated/rubert-tiny2")
rubert_tiny_model = TransformerClassificationModel(brt_base, 7).to(DEVICE)
draw_first_layer_attention_maps(text_loader=train_dataset, model=rubert_tiny_model, tokenizer=tokenizer_rt)
```

```
Голов Attention = 12, ниже отрисовка 40.0% из них
```




Head number 2 - attention map
tokens y
[CLS]
[CLS]
graph
shows
graph
##s
three
square
members
numbers
selected
b
c
graph
##s
three
members
a
##x
b
##x
c
b
##x
c
##x
c
##x
a
##x
a
##x
a
##x
b
[SEP]
tokens x
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1


Head number 3 - attention map
tokens y
[CLS]
[CLS]
graph
shows
graph
##s
three
square
members
numbers
selected
b
c
graph
##s
three
members
a
##x
b
##x
c
b
##x
c
##x
c
##x
a
##x
a
##x
a
##x
b
[SEP]
tokens x
0.25
0.20
0.15
0.10
0.05



In [254]:
```
mathbert = BertModel.from_pretrained('tbs17/MathBert')
mathbert_model = TransformerClassificationModel(mathbert, 7).to(DEVICE)
attention, tokens = plot_attention(text_loader=train_dataset, model=mathbert_model)
model_view(attention, tokens)
```

```
<IPython.core.display.Javascript object>
```

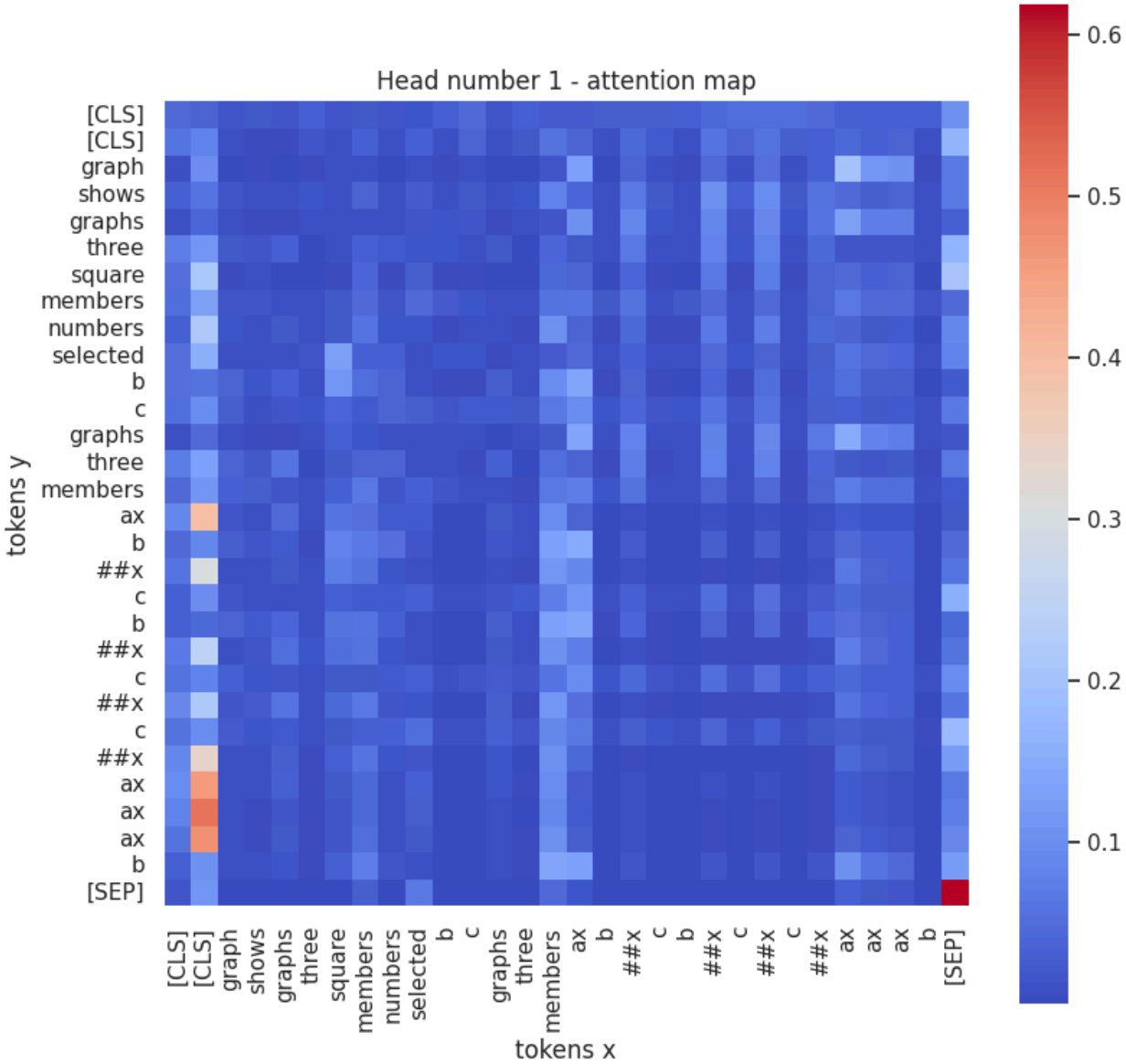
In [255]:
```
head_view(attention, tokens)
```

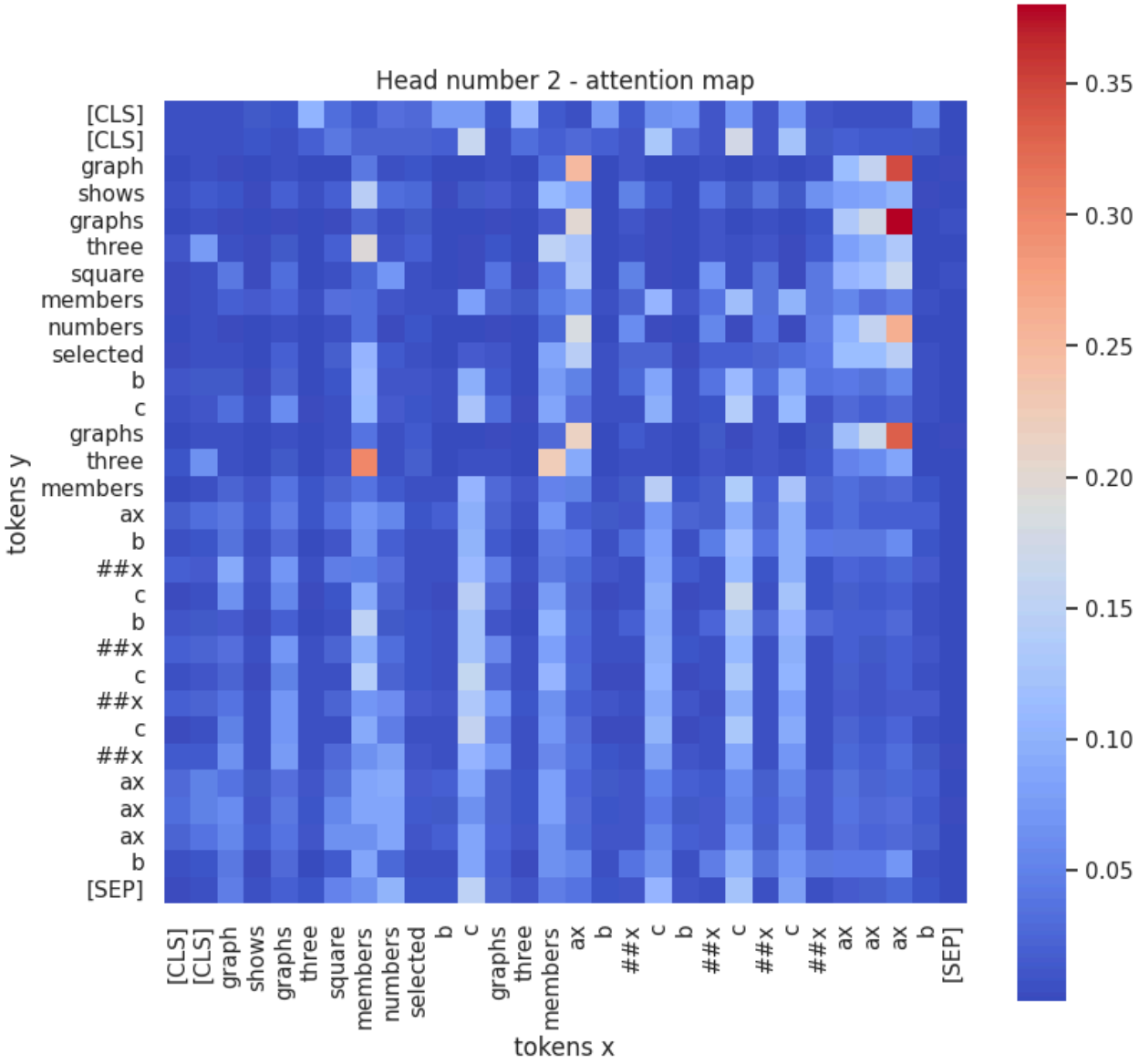
Layer: 

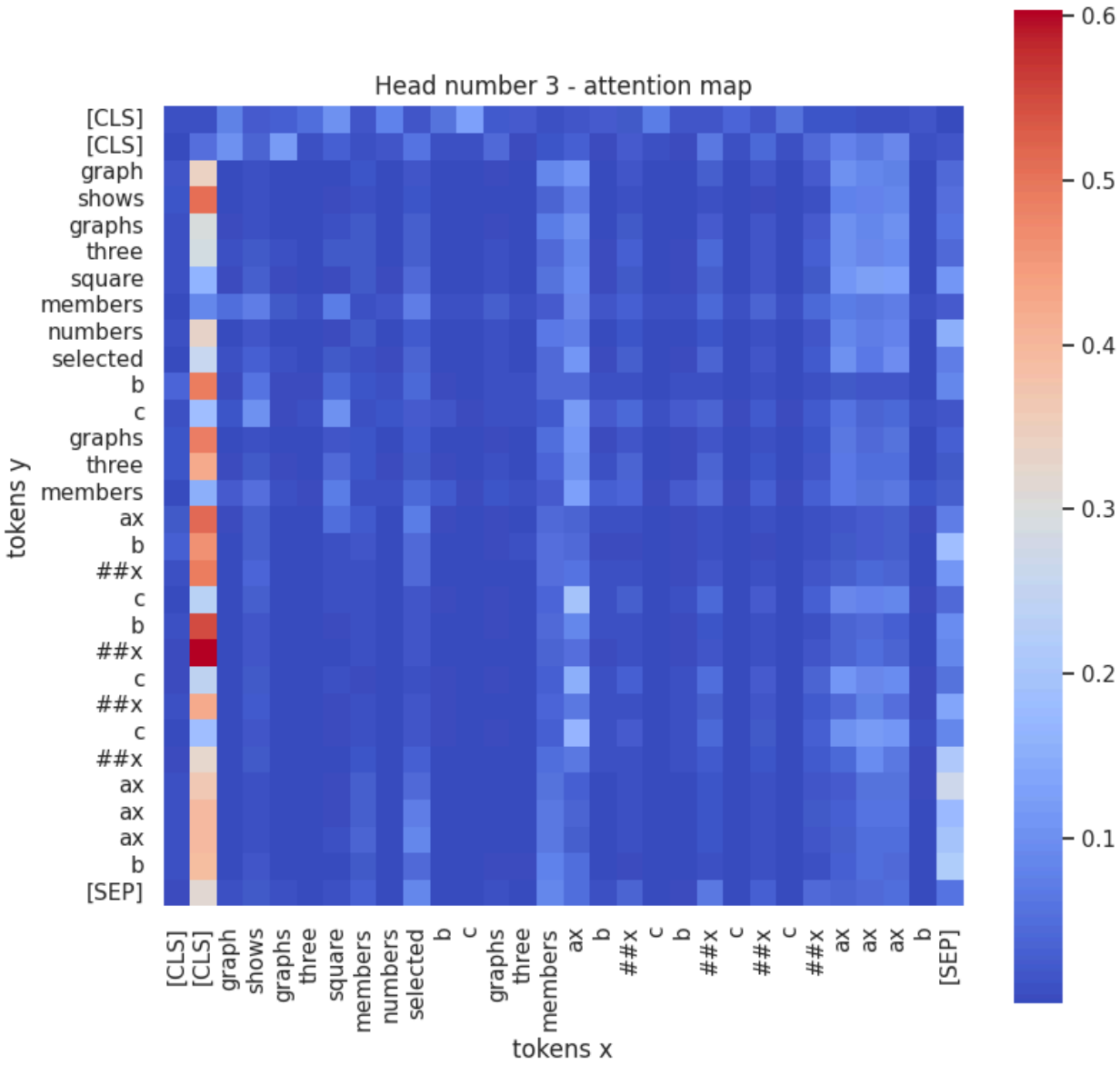
```
<IPython.core.display.Javascript object>
```

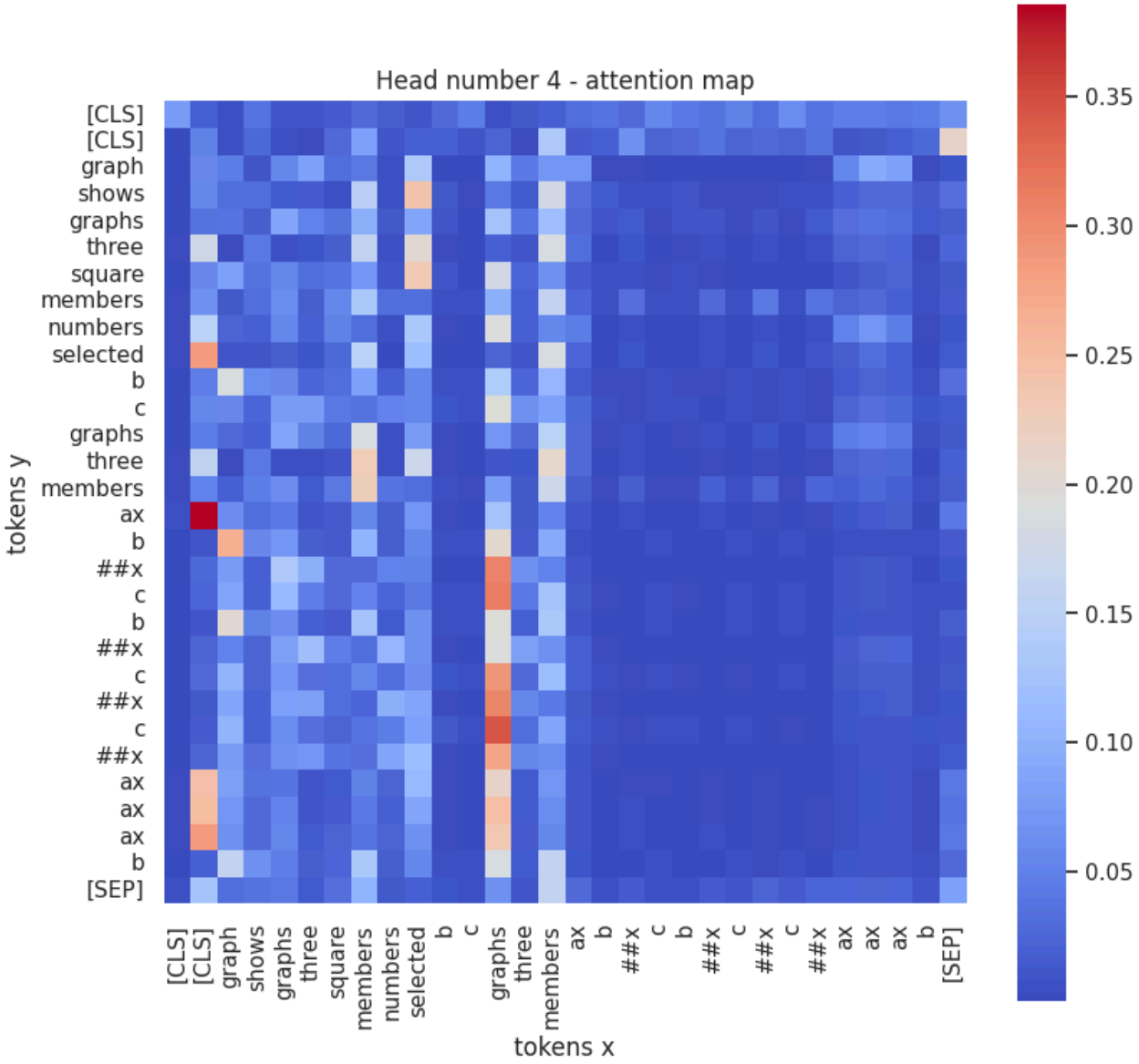
In [256]:
```
draw_first_layer_attention_maps(text_loader=train_dataset, model=mathbert_model, tokenizer=tokenizer_mb)
```

```
Голов Attention = 12, ниже отрисовка 40.0% из них
```




Head number 2 - attention map
tokens y
[CLS]
[CLS]
graph
shows
graphs
three
square
members
numbers
selected
b
c
graphs
three
members
ax
b
##x
c
b
##x
c
b
##x
c
##x
c
##x
ax
ax
ax
b
[SEP]
tokens x
[CLS]
[CLS]
graph
shows
graphs
three
square
members
numbers
selected
b
c
graphs
three
members
ax
b
##x
c
b
##x
c
##x
c
##x
ax
ax
ax
b
[SEP]
0.35
0.30
0.25
0.20
0.15
0.10
0.05


Head number 3 - attention map
tokens y
[CLS]
[CLS]
graph
shows
graphs
three
square
members
numbers
selected
b
c
graphs
three
members
ax
b
##x
c
b
##x
c
##x
c
##x
ax
ax
ax
b
[SEP]
tokens x
[CLS]
[CLS]
graph
shows
graphs
three
square
members
numbers
selected
b
c
graphs
three
members
ax
b
##x
c
b
##x
c
##x
c
##x
ax
ax
ax
b
[SEP]
0.6
0.5
0.4
0.3
0.2
0.1

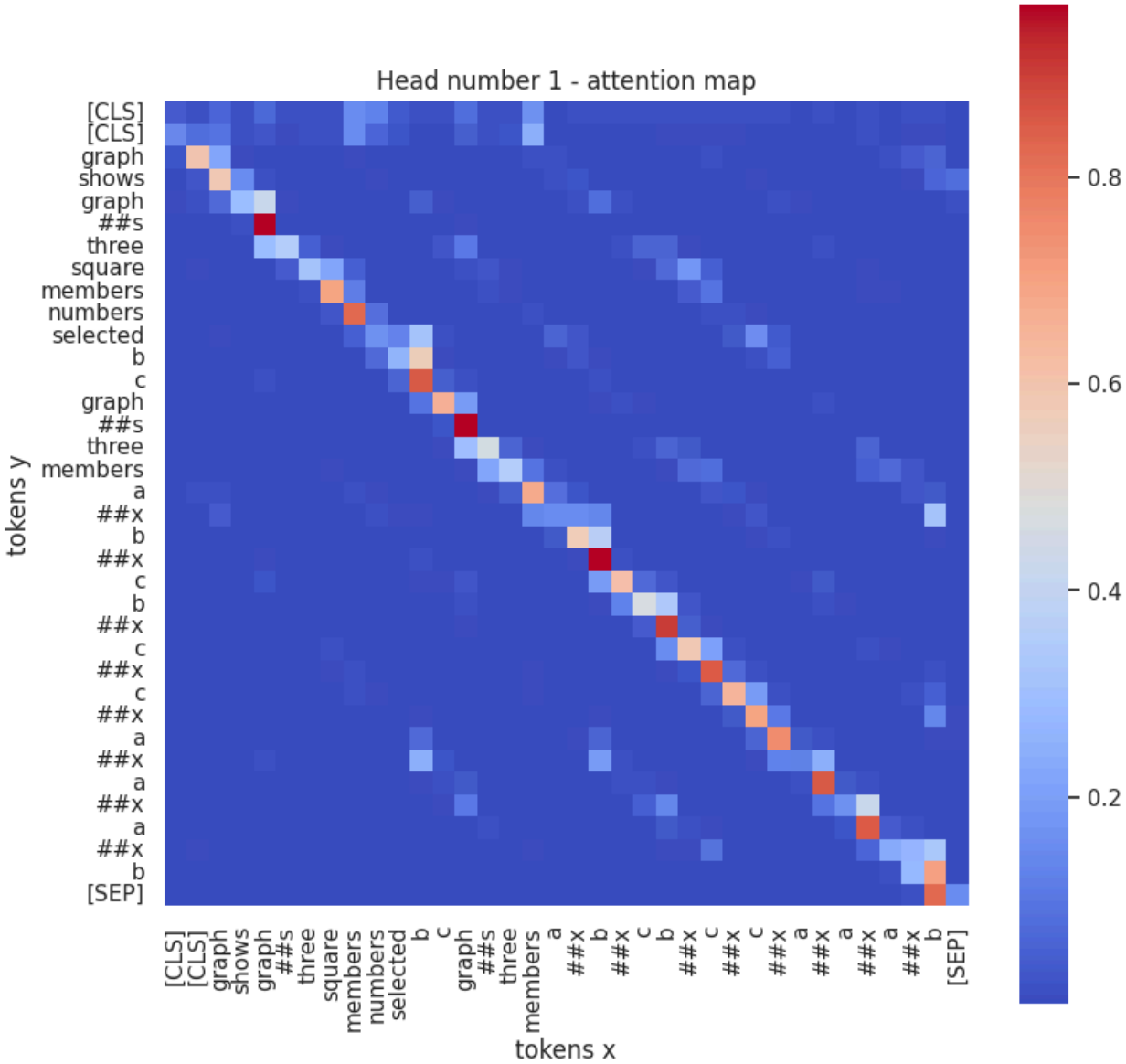


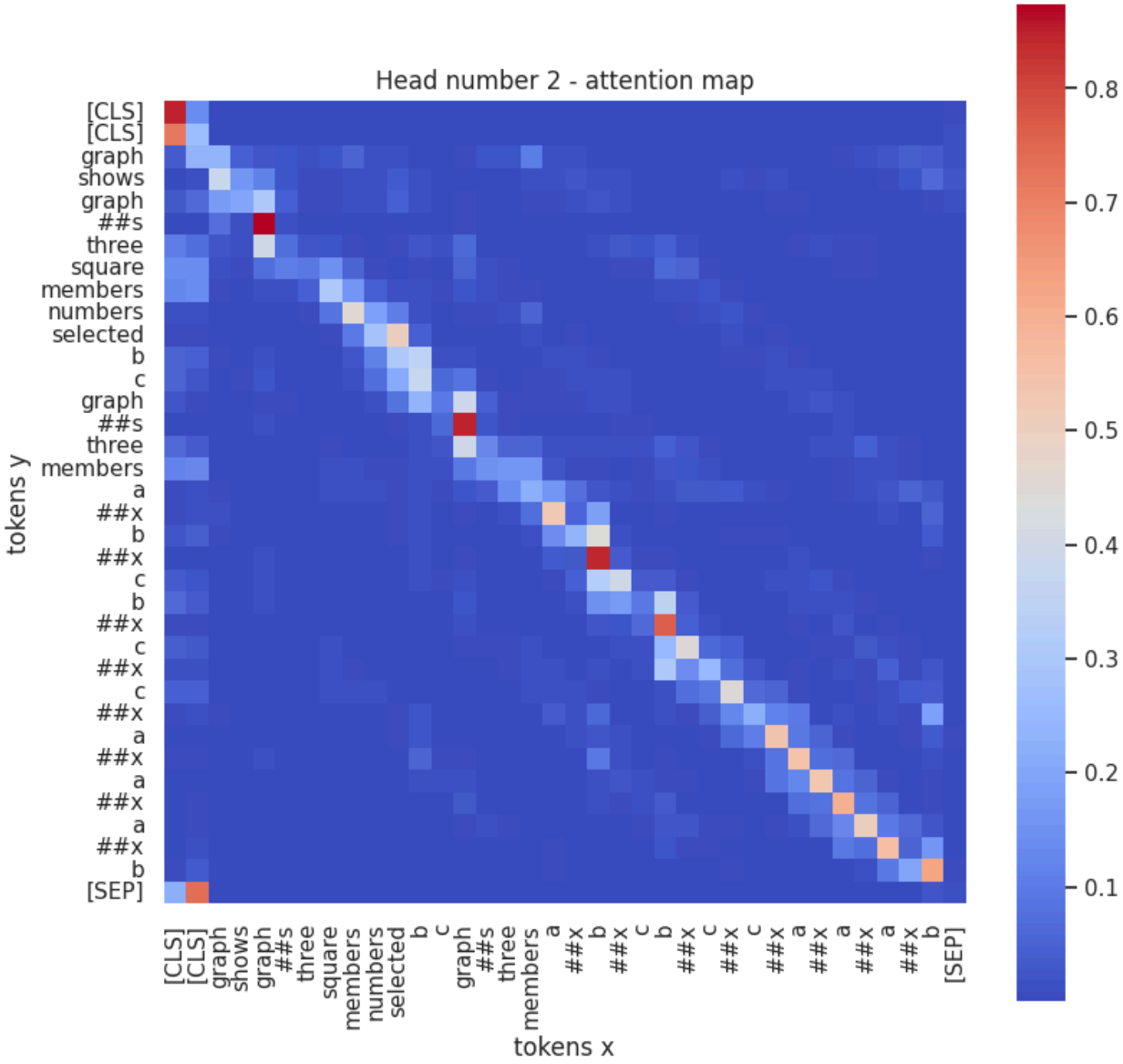
Задание 8 (1 балл)

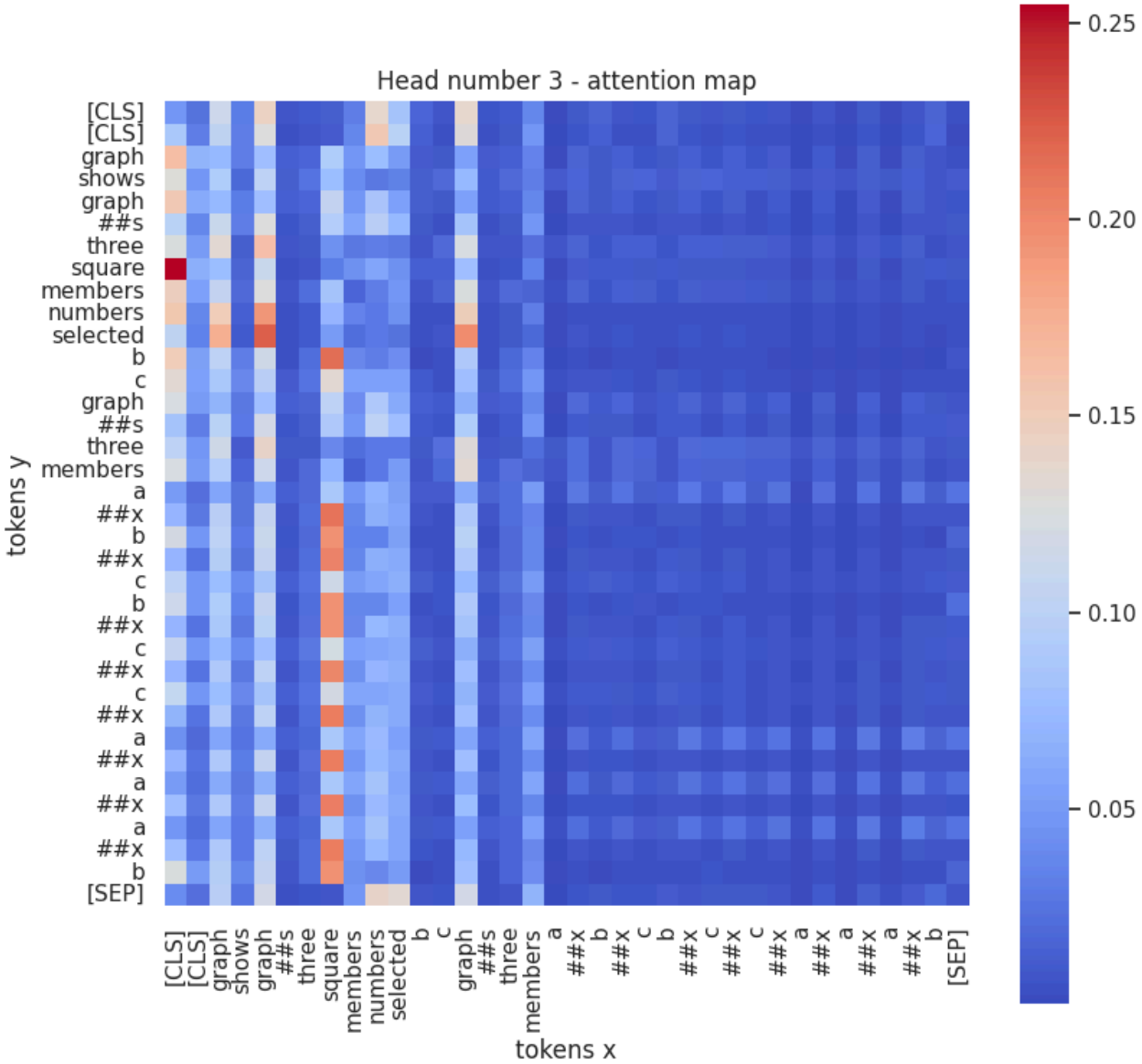
Сделайте то же самое для дообученных моделей. Изменились ли карты внимания и связи, которые они улавливают? Почему?

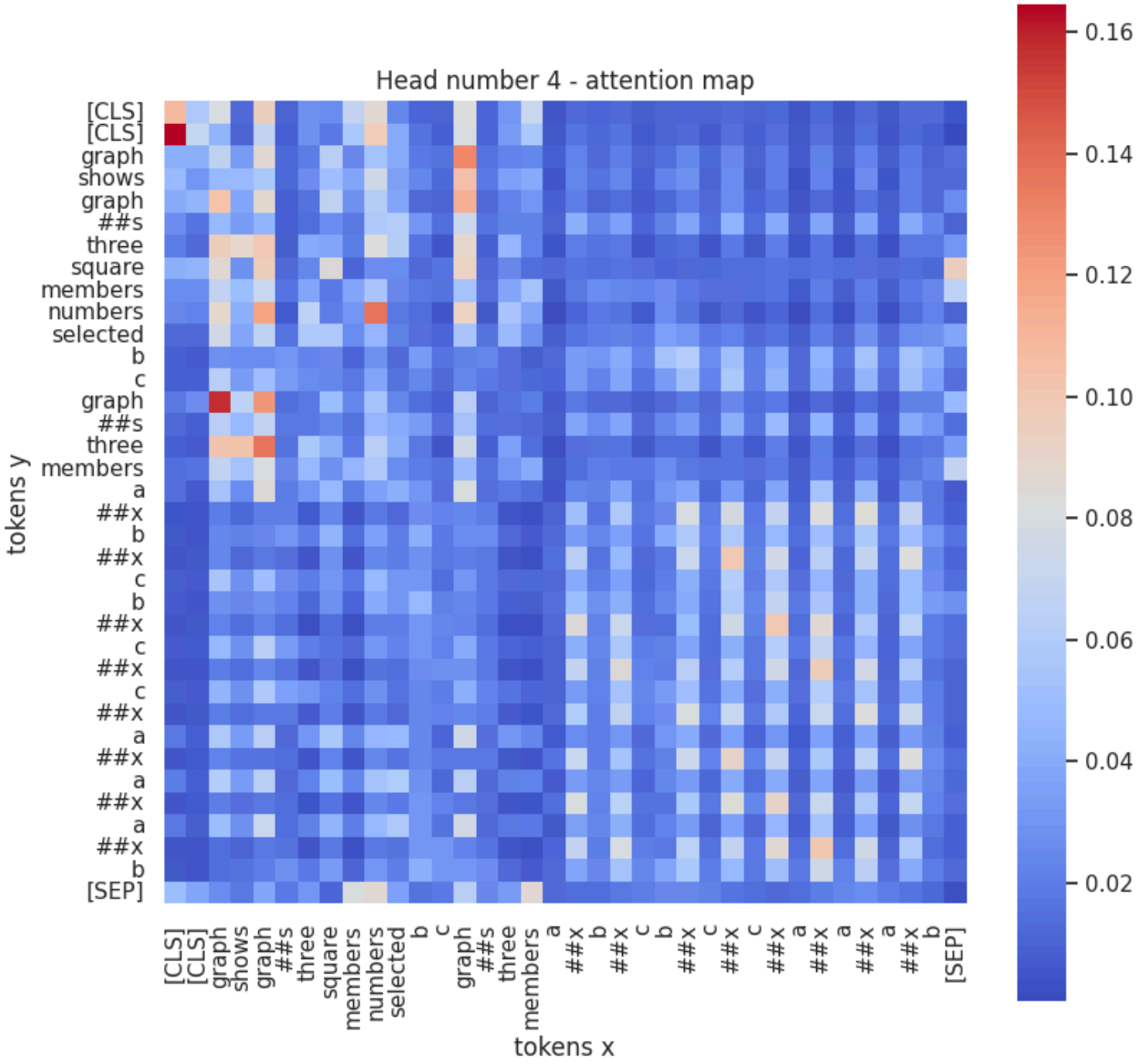
In [74]:
```
draw_first_layer_attention_maps(text_loader=train_dataset, model=rubert_tiny_unfrz, tokenizer=tokenizer_rt)
```

```
Голов Attention = 12, ниже отрисовка 40.0% из них
```




Head number 2 - attention map
tokens y
[CLS]
[CLS]
graph
shows
graph
##s
three
square
members
numbers
selected
b
c
graph
##s
three
members
a
##x
b
##x
c
b
##x
c
##x
c
##x
a
##x
a
##x
a
##x
b
[SEP]
tokens x
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1


Head number 3 - attention map
tokens y
[CLS]
[CLS]
graph
shows
graph
##s
three
square
members
numbers
selected
b
c
graph
##s
three
members
a
##x
b
##x
c
b
##x
c
##x
c
##x
a
##x
a
##x
a
##x
b
[SEP]
tokens x
0.25
0.20
0.15
0.10
0.05



In [258]:
```
model_view(attention, tokens)
```

```
<IPython.core.display.Javascript object>
```

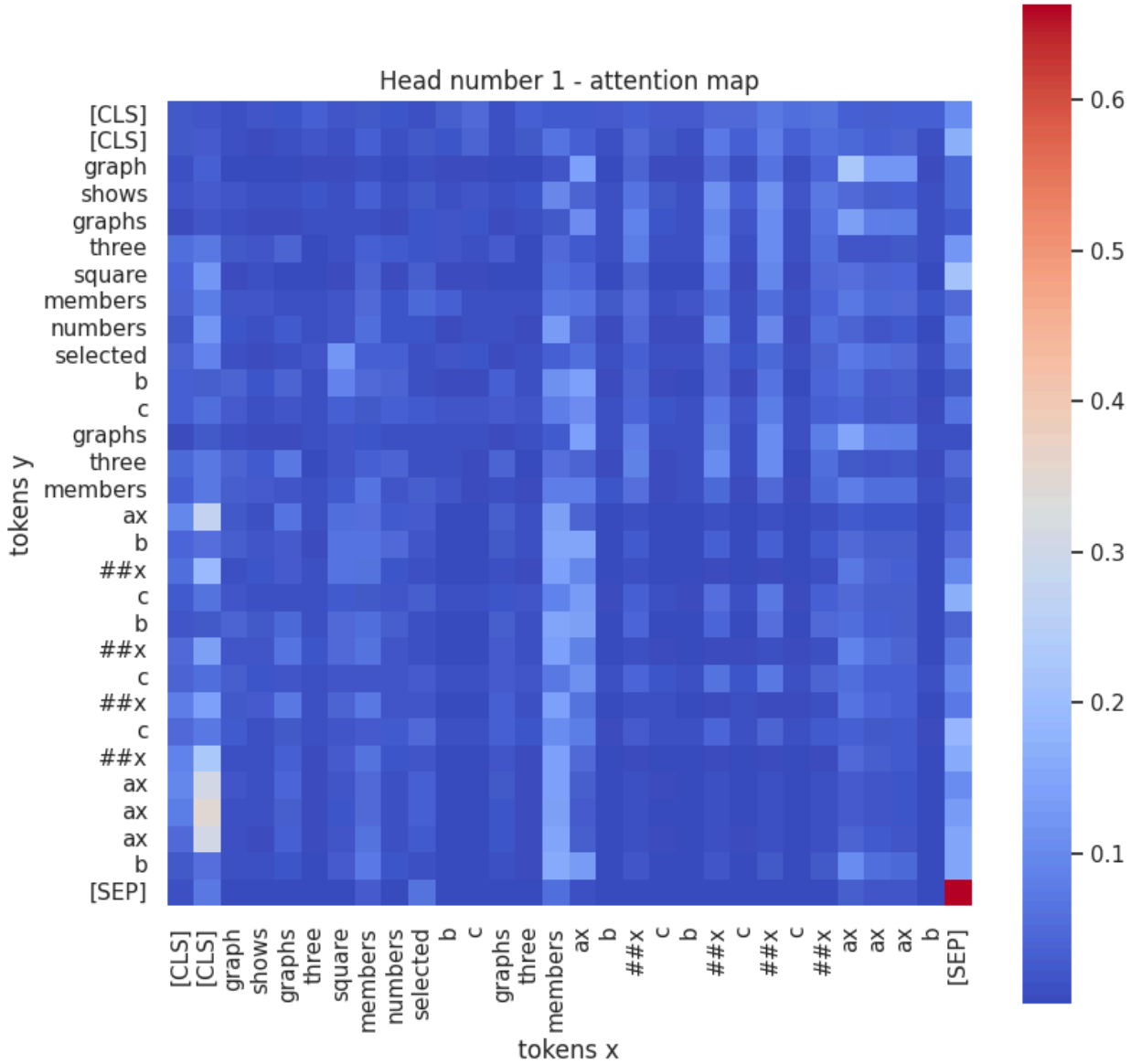
In [257]:
```
attention, tokens = plot_attention(text_loader=train_dataset, model=mathbert_unfrz)
head_view(attention, tokens)
```

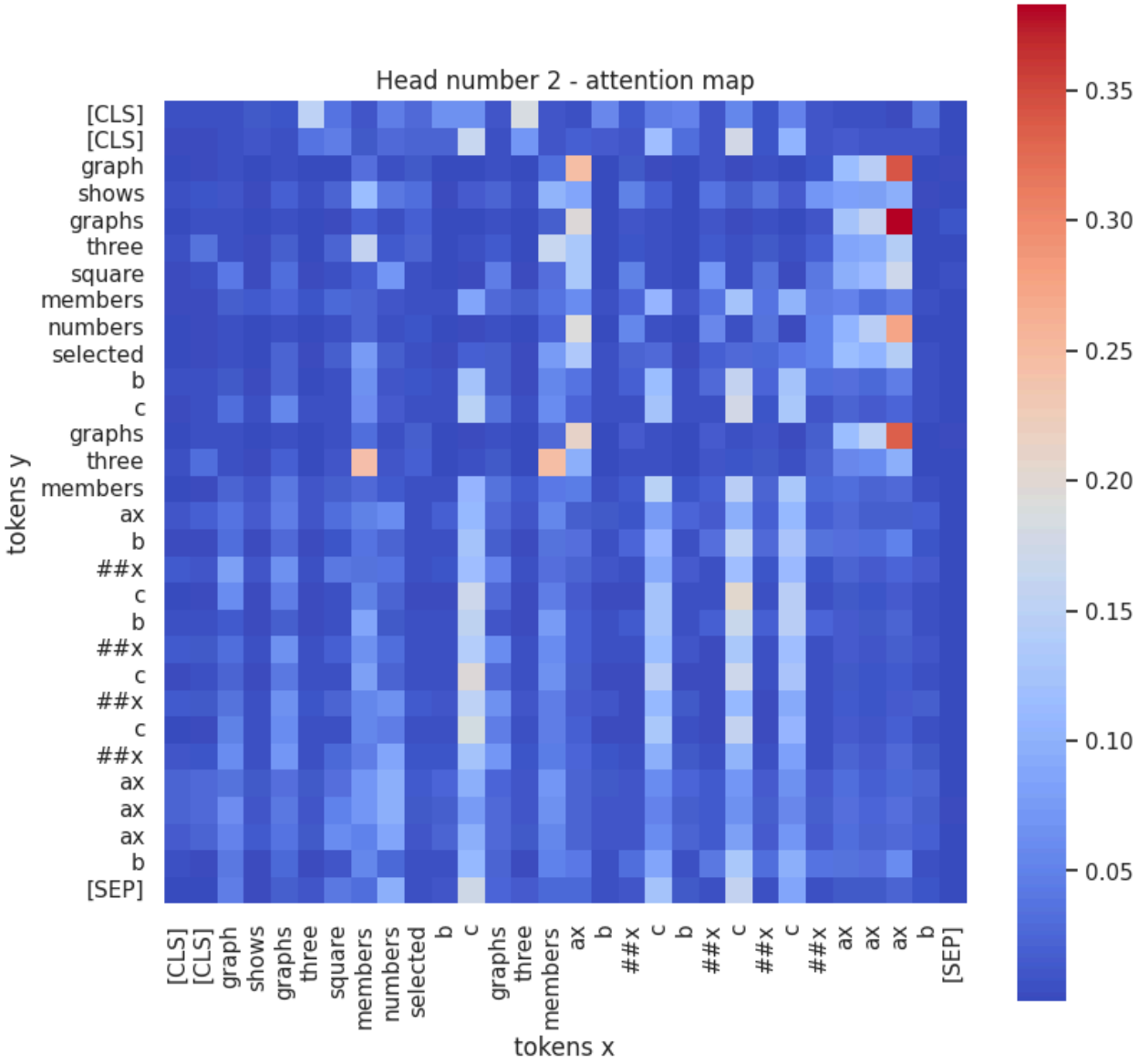
Layer:

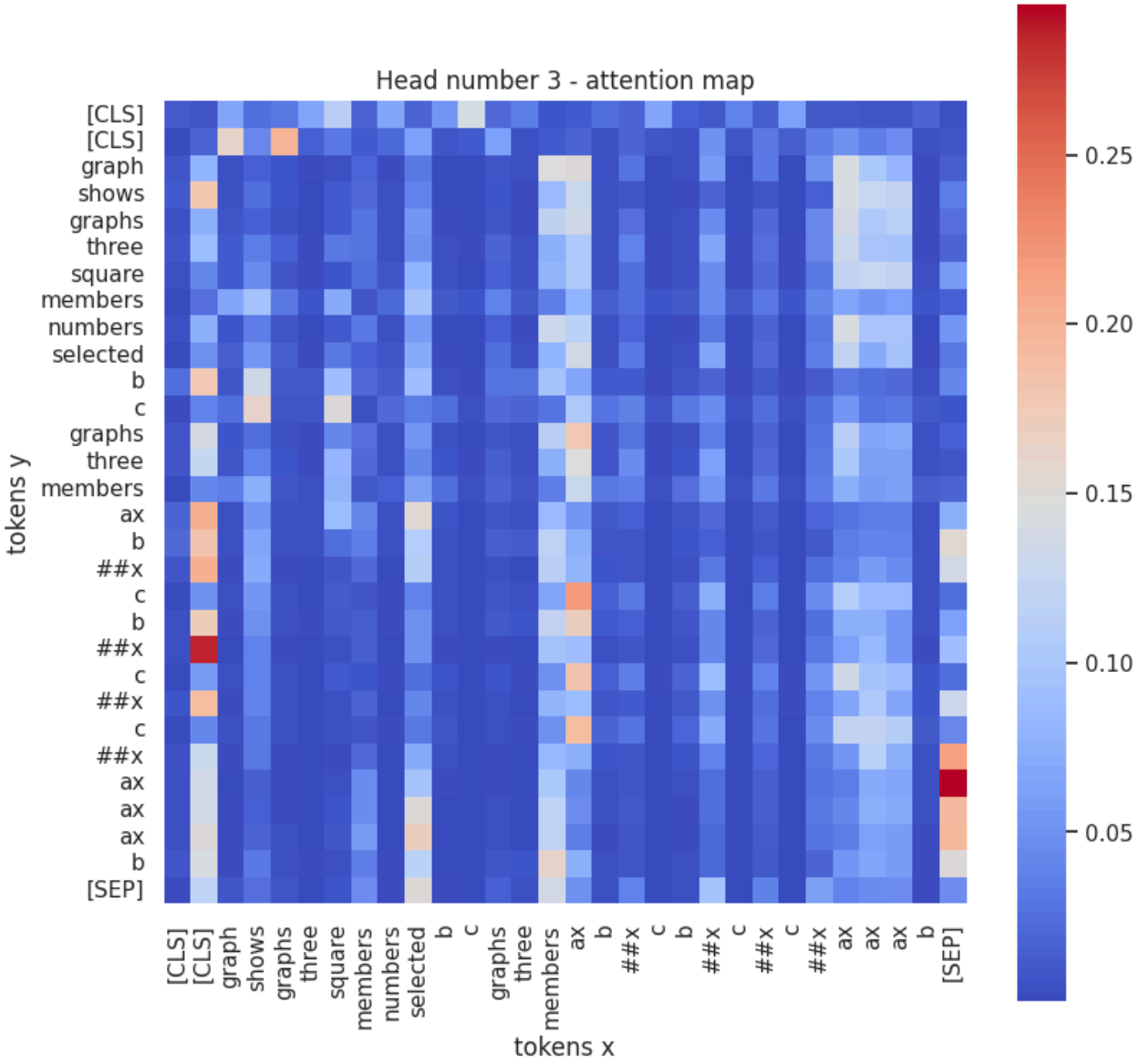
```
<IPython.core.display.Javascript object>
```

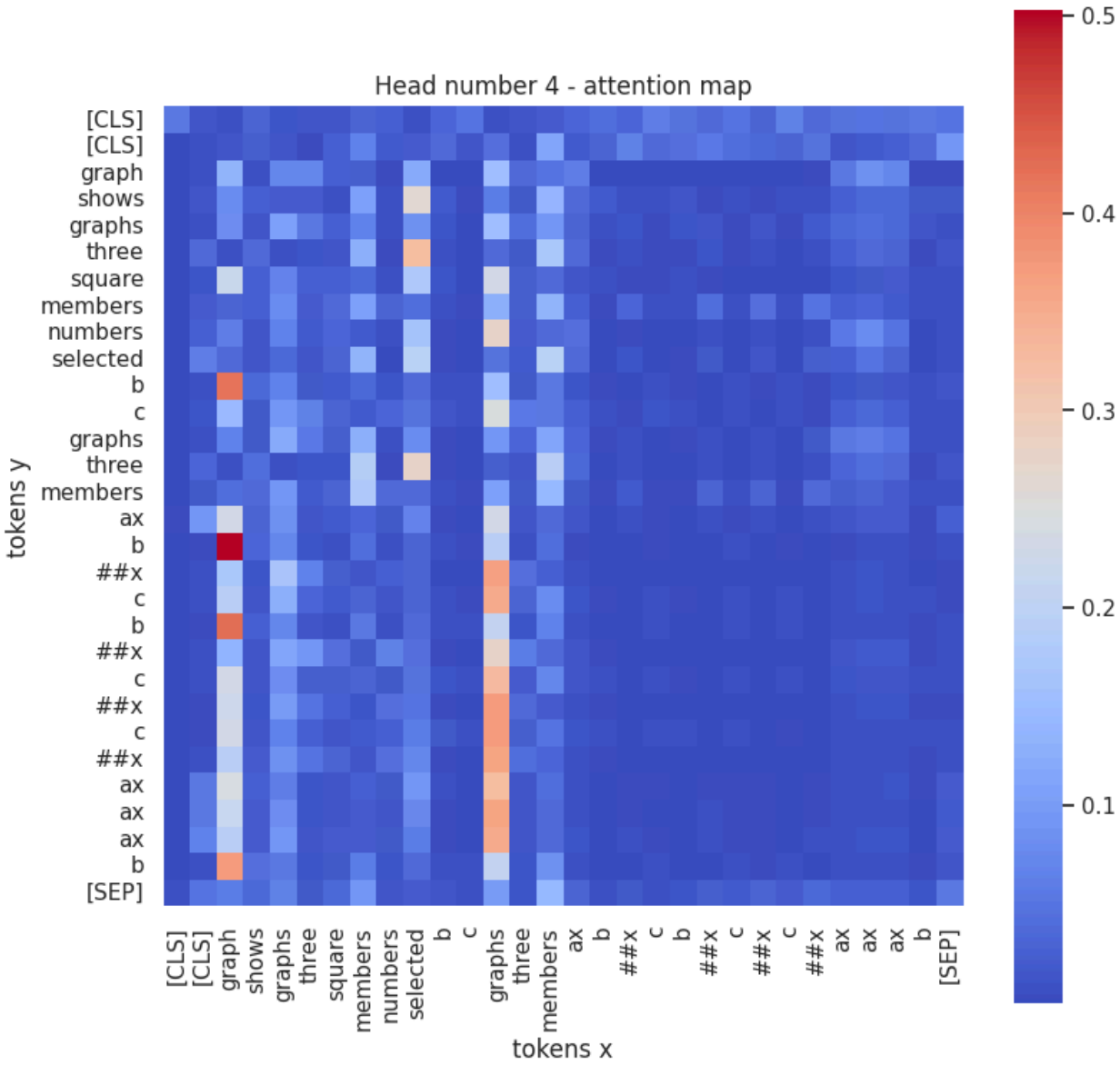
In [75]:
```
draw_first_layer_attention_maps(text_loader=train_dataset, model=mathbert_unfrz, tokenizer=tokenizer_mb)
```

```
Голов Attention = 12, ниже отрисовка 40.0% из них
```




Head number 2 - attention map
tokens y
[CLS]
[CLS]
graph
shows
graphs
three
square
members
numbers
selected
b
c
graphs
three
members
ax
b
##x
c
b
##x
c
b
##x
c
##x
c
##x
ax
ax
ax
b
[SEP]
tokens x
0.35
0.30
0.25
0.20
0.15
0.10
0.05


Head number 3 - attention map
tokens y
[CLS]
[CLS]
graph
shows
graphs
three
square
members
numbers
selected
b
c
graphs
three
members
ax
b
##x
c
b
##x
c
##x
c
##x
ax
ax
ax
b
[SEP]
tokens x
[CLS]
[CLS]
graph
shows
graphs
three
square
members
numbers
selected
b
c
graphs
three
members
ax
b
##x
c
b
##x
c
##x
c
##x
ax
ax
ax
b
[SEP]
0.25
0.20
0.15
0.10
0.05



Выводы по картам внимания:

1. **Карты внимания в моделях rubert-tiny2 и MathBert хорошо показывают их содержательную разницу**:

- если неспец rubert-tiny2 "концентрирует внимание" токенов на самих себя, либо на ближайших "соседях", что соответствует выраженным диагональным линиям на диаграммах,
- в MathBert вообще не присутствуют диагональные яркие линии, при этом есть много вертикальных, что означает выявленные семантические связи токенов друг с другом, для некоторых токенов - по всему тексту.

2. **Также обращает на себя внимание, что attention в MathBert смог выявить связи по всему контексту для ключевых слов**: graph, square, members. graph, graphs MathBert с размороженным" последним слоем - ключевые слова, что соответствует одному из классов в датасете, что говорит об адекватности модели.
3. **В целом, нет сильных структурных отличий между одними и теми же моделями**: с "заморозкой" последних слоев и без нее. Результат дообучения модели проявляется. главным образом, в более интенсивной яркости и большей площади покрытия, соответствующих вниманию "голов" при обработке токенов.

In [ ]: