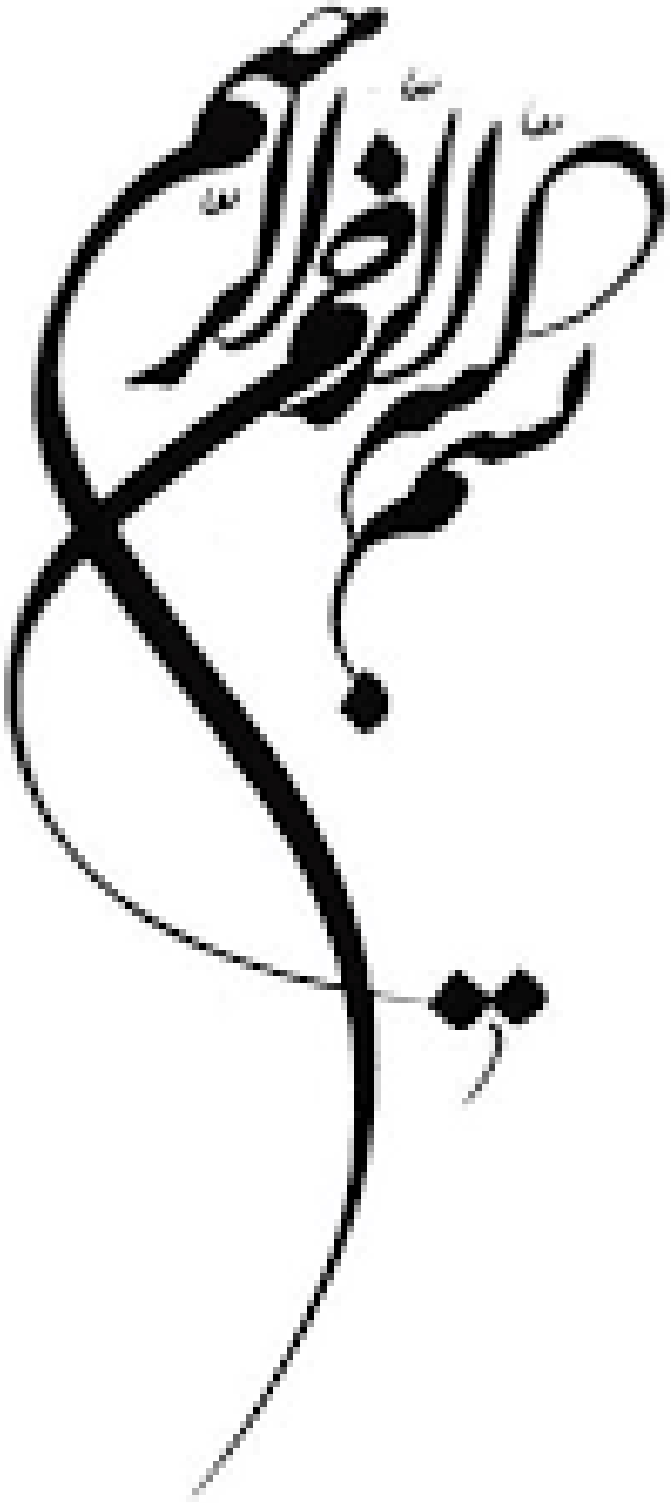


دانشکده برق و سایبرنیک و جنگال

عنوان:

پیاده‌سازی توجه چندسر و نقطه‌ای

استاد:

دکتر سدید پور

نگارش:

فاطمه آقائی

آبان ۱۴۰۲

فهرست مطالب

عنوان	صفحه

فهرست شکل‌ها

شماره شکل	عنوان شکل	صفحه

۱ تشریح مسئله

در این پروژه در ارتباط با مکانیزم توجه و پیاده‌سازی آن بر روی کلمات واقعی تحقیق شده است.

۲ برداری کردن کلمات واقعی با استفاده از مدل زبانی برت

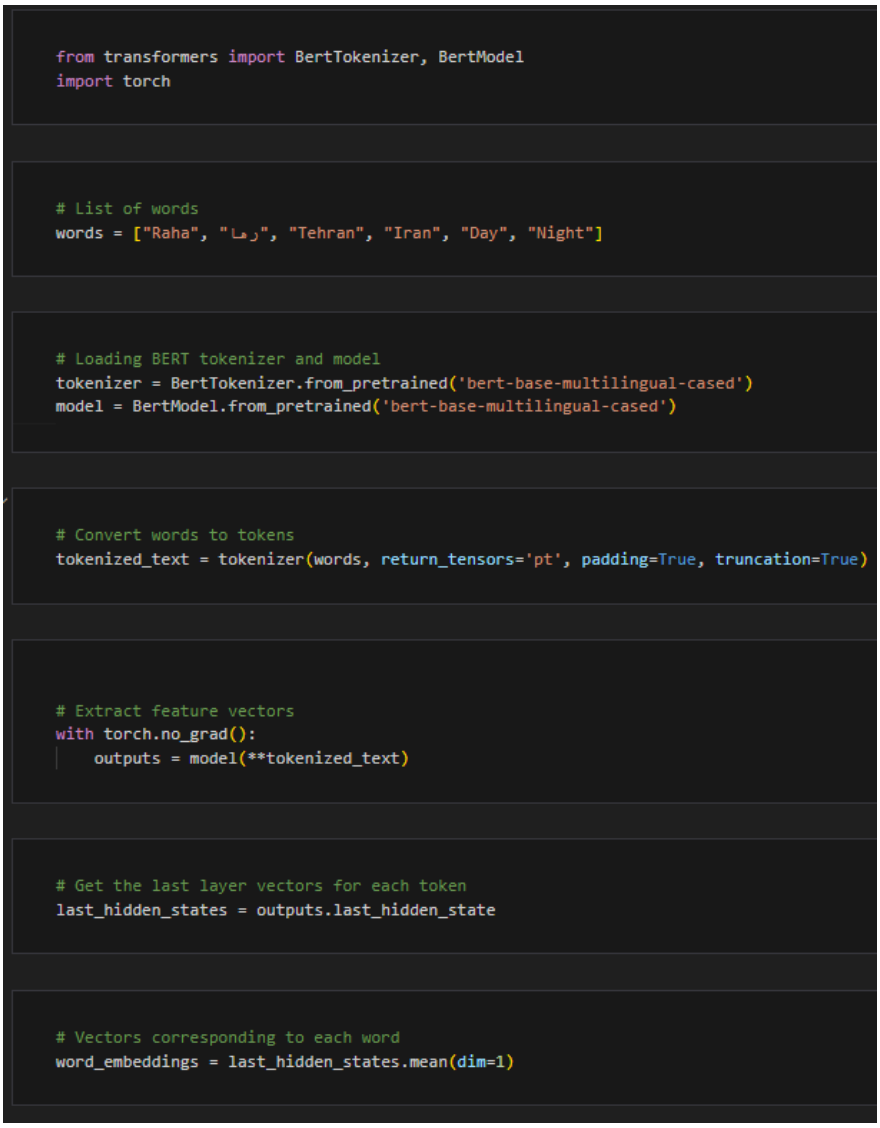
ابتدا کتابخانه‌های مورد نیاز برای استفاده از مدل BERT، یعنی BertTokenizer و BertModel از کتابخانه transformers و کتابخانه torch برای مدیریت تنسورها (Tensor) و اجرای محاسبات با PyTorch فراخوانی شده‌اند. سپس یک لیست از واژگان به نام words تعریف شده است که شامل چند واژه به زبان انگلیسی و فارسی است. توکن‌ساز و مدل BERT با استفاده از توابع BertTokenizer.from_pretrained و BertModel.from_pretrained از کتابخانه transformers بارگیری شده‌اند. در مرحله بعد تبدیل واژگان به توکن‌ها انجام شده است. برای این کار تابع tokenizer برای تبدیل لیست words به توکن‌ها فراخوانی شده است. توکن‌ها به صورت توابع PyTorch تنسور (tensor) با توجه به پارامترهای return_tensors='pt'، padding=True و truncation=True تولید شده‌اند. توضیح این پارامترها به صورت زیر می‌باشد:

۱) $return\_tensors = 'pt'$: این پارامتر نحوه بازگشت خروجی تابع tokenizer را تعیین می‌کند. در اینجا، 'pt' به معنای PyTorch است، بنابراین خروجی به صورت تنسور PyTorch ارائه می‌شود.

۲) $padding = True$: اگر این پارامتر فعال باشد، توکن‌ها به گونه‌ای پر می‌شوند که اندازه همگرا به اندازه بزرگترین توکن در دسته (batch) شود. این کار برای ایجاد یک ماتریس تنسور با ابعاد یکسان برای ورودی مدل لازم است.

۳) $truncation = True$: این پارامتر به معنای این است که اگر توکن‌ها از اندازه مشخصی بزرگتر باشند، آن‌ها به اندازه مشخصی کوچکتر می‌شوند. این کار برای اطمینان از اینکه داده‌ها به اندازه‌ای مناسب برای ورود به مدل باشند انجام می‌شود.

با استفاده از این پارامترها، تابع tokenizer توکن‌های متن را به یک فرمت مناسب برای ورود به مدل BERT تبدیل کرده و در tokenized_text ذخیره کرده است. با استفاده از مدل BERT، بردارهای ویژگی برای توکن‌ها به دست آمده است. برای استخراج بردارهای آخرین لایه برای هر توکن، بردارهای مربوط به هر توکن از آخرین لایه استخراج شده و در last_hidden_states ذخیره شده است. با میانگین‌گیری از بردارهای مربوط به هر توکن، بردارهای متناظر با هر کلمه به دست آمده است. کد انجام این مراحل در شکل ۱-۲ آورده شده است.



```python
from transformers import BertTokenizer, BertModel
import torch


# List of words
words = ["Raha", "رها", "Tehran", "Iran", "Day", "Night"]


# Loading BERT tokenizer and model
tokenizer = BertTokenizer.from_pretrained('bert-base-multilingual-cased')
model = BertModel.from_pretrained('bert-base-multilingual-cased')


# Convert words to tokens
tokenized_text = tokenizer(words, return_tensors='pt', padding=True, truncation=True)


# Extract feature vectors
with torch.no_grad():
    outputs = model(**tokenized_text)


# Get the last layer vectors for each token
last_hidden_states = outputs.last_hidden_state


# Vectors corresponding to each word
word_embeddings = last_hidden_states.mean(dim=1)
```

شکل ۱-۲: کد برداری کردن کلمات واقعی

بردارهای متناظر با هر کلمه در اینجا چاپ شده‌اند که در شکل ۲-۲ قابل مشاهده‌اند.

```
# Print vectors corresponding to each word
for i, word in enumerate(words):
    print(f"{word}: {word_embeddings[i]}")
```

```
Raha: tensor([ 6.1210e-01, -2.5609e-01,  4.4647e-01,  4.1287e-01,  3.0716e-01,
        -3.7892e-01, -2.7090e-01,  3.9372e-02,  3.6197e-01,  3.1274e-01,
        -1.7626e-01,  1.5327e-01, -9.2002e-02,  2.2171e-01, -4.5700e-01,
         1.5121e-01,  1.6403e-01,  1.4905e-01, -2.6432e-01,  6.0706e-01,
        -2.0920e-01,  8.7376e-02, -1.0021e+00,  2.1142e-01,  5.0281e-01,
        -8.3765e-01, -3.5324e-01,  6.0821e-01, -3.3136e-01,  5.8034e-01,
        -2.4704e-01,  5.9326e-01, -9.0345e-02, -5.9042e-01,  3.2147e-02,
        -1.8098e-01, -6.4916e-01,  3.6035e-01,  3.1779e-01, -2.5249e-01,
        -1.3084e-02, -2.3081e-01, -5.1205e-01,  3.6903e-01, -1.9387e-01,
        -5.9683e-01, -4.8542e-01, -3.1434e-01, -3.8326e-01, -1.0760e+00,
         2.3769e-02, -1.1365e-01,  2.6216e-01, -3.1834e-01,  3.6565e-01,
        -1.1890e-01,  3.5344e-01, -3.6404e-01, -5.6781e-01, -1.6241e-01,
         2.0685e-02,  3.5285e-01,  1.0186e-01, -8.7363e-02, -6.3310e-01,
        -5.9807e-01,  2.6873e-02,  2.3222e-01, -4.2639e-01,  6.8701e-02,
         4.5650e-01,  8.3176e-02,  2.6063e-01,  5.2804e-02, -4.4717e-01,
         4.4619e-03,  1.3303e-01, -3.3777e-01, -6.6044e-01,  2.4975e-02,
         5.5657e-01,  1.5406e-01, -5.7197e-01,  3.4464e-01, -7.1561e-02,
         5.2873e-01,  5.2537e-01, -4.8853e-01, -9.2953e-02, -1.0304e+00,
         2.3930e-01,  3.5742e-01,  6.6581e-02,  1.8289e-01, -2.7901e-01,
         6.4265e-01, -1.7045e-04,  3.0713e-01,  4.0329e-01,  5.0541e-01,
         4.7118e-01,  3.0178e-01,  5.2200e-01,  6.7111e-02,  3.5795e-01,
         4.6345e-01,  8.2476e-01, -3.5695e-01,  8.2073e-02,  2.8926e-01,
         4.6703e-01,  6.1811e-01, -2.7546e-01, -7.6753e-02, -6.1912e-01,
         7.0387e-02, -7.4453e-01,  7.5072e-02, -2.0868e-01,  1.3186e-01,
        -1.6998e-01,  6.9806e-02,  1.1472e-01, -5.7328e-01,  2.5237e-01,
...
        -3.2164e-01, -7.4346e-01,  2.1041e-01,  2.5009e-02,  3.7196e-01,
         3.6555e-01, -1.3131e-01, -1.9533e-01,  2.4158e-01,  1.7435e-01,
        -3.7378e-01,  1.0634e-01,  5.5050e-01,  5.5823e-01, -9.7981e-02,
         4.3023e-01, -1.2843e-01,  3.5162e-01])
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

شکل ۲-۲: نمایش بردارهای متناظر با هر کلمه

برای نمایش بردارها، بردارهای مربوط به هر کلمه به عنوان یک متغیر به نام word_i ذخیره شده و سپس چاپ شده‌اند. خروجی آن در شکل ۳-۲ قابل مشاهده است.

```
# Displaying the embeddings
for i, word in enumerate(words):
    variable_name = f"word_{i+1}"
    globals()[variable_name] = word_embeddings[i]
    print(f"{variable_name}: {word_embeddings[i]}")
```

```
word_1: tensor([ 6.1210e-01, -2.5609e-01,  4.4647e-01,  4.1287e-01,  3.0716e-01,
        -3.7892e-01, -2.7090e-01,  3.9372e-02,  3.6197e-01,  3.1274e-01,
        -1.7626e-01,  1.5327e-01, -9.2002e-02,  2.2171e-01, -4.5700e-01,
         1.5121e-01,  1.6403e-01,  1.4905e-01, -2.6432e-01,  6.0706e-01,
        -2.0920e-01,  8.7376e-02, -1.0021e+00,  2.1142e-01,  5.0281e-01,
        -8.3765e-01, -3.5324e-01,  6.0821e-01, -3.3136e-01,  5.8034e-01,
        -2.4704e-01,  5.9326e-01, -9.0345e-02, -5.9042e-01,  3.2147e-02,
        -1.8098e-01, -6.4916e-01,  3.6035e-01,  3.1779e-01, -2.5249e-01,
        -1.3084e-02, -2.3081e-01, -5.1205e-01,  3.6903e-01, -1.9387e-01,
        -5.9683e-01, -4.8542e-01, -3.1434e-01, -3.8326e-01, -1.0760e+00,
         2.3769e-02, -1.1365e-01,  2.6216e-01, -3.1834e-01,  3.6565e-01,
        -1.1890e-01,  3.5344e-01, -3.6404e-01, -5.6781e-01, -1.6241e-01,
         2.0685e-02,  3.5285e-01,  1.0186e-01, -8.7363e-02, -6.3310e-01,
        -5.9807e-01,  2.6873e-02,  2.3222e-01, -4.2639e-01,  6.8701e-02,
         4.5650e-01,  8.3176e-02,  2.6063e-01,  5.2804e-02, -4.4717e-01,
         4.4619e-03,  1.3303e-01, -3.3777e-01, -6.6044e-01,  2.4975e-02,
         5.5657e-01,  1.5406e-01, -5.7197e-01,  3.4464e-01, -7.1561e-02,
         5.2873e-01,  5.2537e-01, -4.8853e-01, -9.2953e-02, -1.0304e+00,
         2.3930e-01,  3.5742e-01,  6.6581e-02,  1.8289e-01, -2.7901e-01,
         6.4265e-01, -1.7045e-04,  3.0713e-01,  4.0329e-01,  5.0541e-01,
         4.7118e-01,  3.0178e-01,  5.2200e-01,  6.7111e-02,  3.5795e-01,
         4.6345e-01,  8.2476e-01, -3.5695e-01,  8.2073e-02,  2.8926e-01,
         4.6703e-01,  6.1811e-01, -2.7546e-01, -7.6753e-02, -6.1912e-01,
         7.0387e-02, -7.4453e-01,  7.5072e-02, -2.0868e-01,  1.3186e-01,
        -1.6998e-01,  6.9806e-02,  1.1472e-01, -5.7328e-01,  2.5237e-01,
...
        -3.2164e-01, -7.4346e-01,  2.1041e-01,  2.5009e-02,  3.7196e-01,
         3.6555e-01, -1.3131e-01, -1.9533e-01,  2.4158e-01,  1.7435e-01,
        -3.7378e-01,  1.0634e-01,  5.5050e-01,  5.5823e-01, -9.7981e-02,
         4.3023e-01, -1.2843e-01,  3.5162e-01])
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

شکل ۳-۲: نمایش بردارها

۳ طراحی مکانیزم توجه نقطه‌ای با کتابخانه‌ی پایتورچ

در این قسمت محاسبه توجه (attention) با استفاده از مدل‌های ترنسفورمر انجام شده است. برای این کار ابتدا ابعاد بردارها با استفاده از متغیر hidden_size تعیین شده‌اند که در اینجا مقدار ثابت ۷۶۸ برای ابعاد بردارهای خروجی مدل BERT است. سپس با استفاده از تابع torch.randn، بردارهای Q، K، و V (بردارهای Query (Q)، Key (K) و Value (V)) برای هر کلمه به طول hidden_size ایجاد شده است. بردارهای Q، K، و V برای جلوگیری از انفجار گرادیان نرمال‌سازی شده‌اند. امتیازهای توجه با ماتریس ضرب داخلی بین بردارهای Q و K و سپس با استفاده از تابع softmax محاسبه شده‌اند. با استفاده از امتیازهای توجه و بردارهای V، خروجی نهایی با محاسبه میانگین وزن‌دار ایجاد شده‌اند. کد این قسمت در شکل ۱-۳ آورده شده است.

```python
import torch

# Dimensions of vectors
hidden_size = 768  # Dimensions of BERT vectors


# Define Q, K, and V vectors using torch.randn
Q = torch.randn(1, len(words), hidden_size)  # Query vector
K = torch.randn(1, len(words), hidden_size)  # Key vector
V = torch.randn(1, len(words), hidden_size)  # Value vector

# Normalize Q, K, and V vectors to prevent gradient explosion
Q = Q / torch.sqrt(torch.tensor(hidden_size, dtype=torch.float32))
K = K / torch.sqrt(torch.tensor(hidden_size, dtype=torch.float32))
V = V / torch.sqrt(torch.tensor(hidden_size, dtype=torch.float32))


# Calculate attention scores
attention_scores = torch.matmul(Q, K.transpose(-1, -2))
attention_scores = torch.nn.functional.softmax(attention_scores, dim=-1)


# Calculate the weighted average of V vectors based on attention scores
output = torch.matmul(attention_scores, V)
```

شکل ۱-۳: کد مکانیزم توجه نقطه‌ای با کتابخانه‌ی پایتورچ

سپس خروجی نهایی به صورت یک تنسور PyTorch چاپ شده است. این خروجی در شکل ۲-۳ آورده شده است.

```python
# Display the final output
print("Output Tensor:")
print(output)
```

```
Output Tensor:
tensor([[[ 9.4279e-03,  2.6025e-03, -1.2737e-02,  ..., -2.7746e-02,
           8.2943e-05,  1.0699e-02],
         [ 1.0100e-02,  2.4506e-03, -1.1016e-02,  ..., -2.8180e-02,
          -4.5125e-04,  1.0392e-02],
         [ 1.0452e-02,  3.5882e-03, -1.1501e-02,  ..., -2.7960e-02,
          -1.9115e-03,  1.0767e-02],
         [ 1.0493e-02,  2.5342e-03, -1.0263e-02,  ..., -2.7982e-02,
          -9.6794e-04,  1.0366e-02],
         [ 9.7568e-03,  3.5951e-03, -1.2185e-02,  ..., -2.7916e-02,
          -1.2466e-03,  1.0473e-02],
         [ 1.0367e-02,  2.8663e-03, -1.1782e-02,  ..., -2.8578e-02,
          -1.8658e-04,  1.0864e-02]]])
```

شکل ۲-۳: خروجی نهایی توجه نقطه‌ای

برای رسم نمودار حرارتی[1] توجه با استفاده از توابع matplotlib و seaborn یک نمودار حرارتی از ماتریس امتیازهای توجه ترسیم شده است. که می‌توان آن را در شکل ۳-۳ مشاهده نمود.

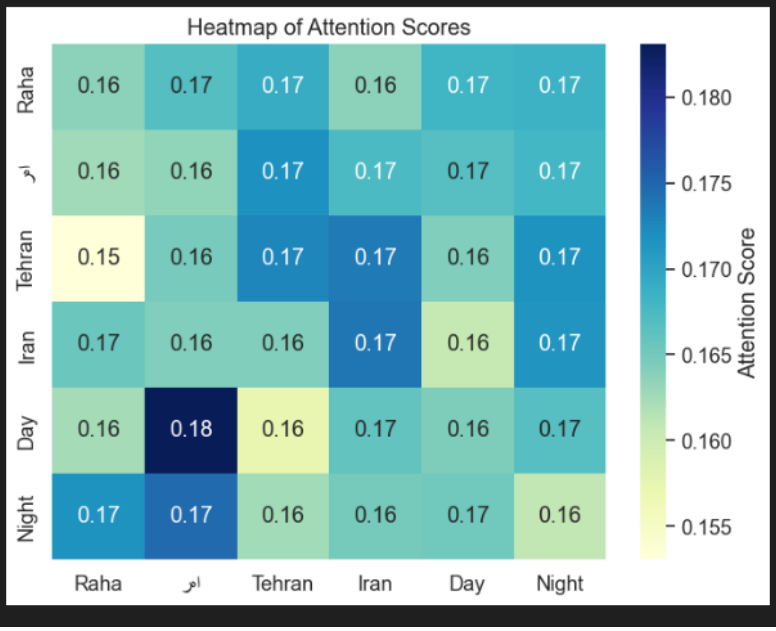
[1] Heat map

```python
import matplotlib.pyplot as plt
import seaborn as sns

# Function to plot the attention heatmap
def plot_attention_heatmap(attention_matrix, words):
    sns.set(font_scale=1.2)
    plt.figure(figsize=(8, 6))
    ax = sns.heatmap(attention_matrix, annot=True, xticklabels=words, yticklabels=words, cmap="YlGnBu", cbar_kws={'label': 'Attention Score'})
    plt.title('Heatmap of Attention Scores')
    plt.show()

# Calculate the weighted average of vectors V based on attention scores
attention_matrix = attention_scores.squeeze().numpy()
plot_attention_heatmap(attention_matrix, words)
```

✓ 4.6s



شکل ۳-۳: نمودار حرارتی ماتریس

در این نمودار به کلمات توجه‌های نزدیک به هم شده است و حتی کلمات یکسان ممکن است توجه کمتری نسبت به کلمات غیریکسان داشته باشند. به همین دلیل یک کد دیگر برای محاسبه توجه زده شده است.

در این کد ابتدا ابعاد بردارها با استفاده از متغیرهای embedding_dim و seq_length تعیین شده‌اند. سپس با استفاده از تابع torch.randn، بردارهای Q، K، و V برای هر کلمه به طول embedding_dim ایجاد شده‌اند. تابع dot_product_attention با استفاده از عملگر ضرب نقطه‌ای، امتیازهای توجه، وزن‌های توجه، و خروجی توجه را محاسبه کرده است. کد این مراحل در شکل ۳-۴ آورده شده است.

```python
import torch

# Dimensions of vectors
embedding_dim = 768
seq_length = len(words)

# Create Q, K and V vectors
Q = torch.randn(seq_length, embedding_dim)
K = torch.randn(seq_length, embedding_dim)
V = torch.randn(seq_length, embedding_dim)
```

```python
Comment Code |
def dot_product_attention(Q, K, V):
    # Compute the inner multiplication scalar between Q and K
    attention_scores = torch.matmul(Q, K.t())

    # Scaling
    attention_scores = attention_scores / torch.sqrt(torch.tensor(embedding_dim, dtype=torch.float32))

    # Apply softmax
    attention_weights = torch.nn.functional.softmax(attention_scores, dim=-1)

    # Calculate the output according to the attention weights
    attention_output = torch.matmul(attention_weights, V)

    return attention_output, attention_weights
```

شکل ۴-۳: کد مراحل محاسبه توجه به روش دوم

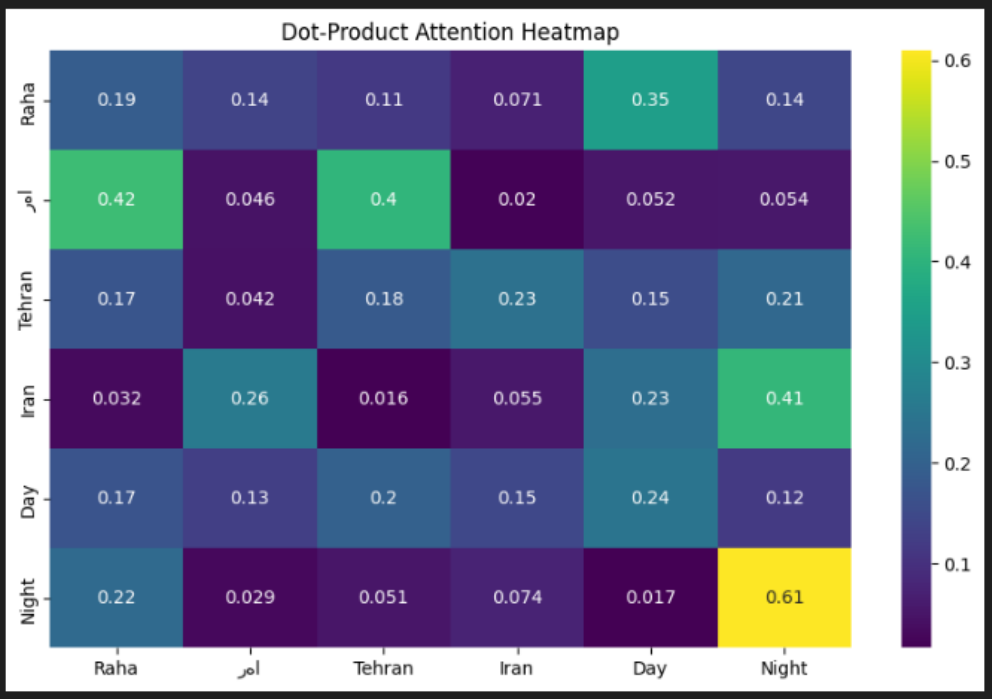
در نهایت با استفاده از تابع dot_product_attention بر روی بردارهای Q، K، و V، امتیازها و وزن‌های توجه محاسبه شده و سپس نقشه حرارتی از این وزن‌ها با استفاده از کتابخانه‌های matplotlib و seaborn رسم شده است که در شکل ۳-۵ قابل مشاهده است.

```python
import matplotlib.pyplot as plt
import seaborn as sns

# Calculation of point attention
attention_output, attention_weights = dot_product_attention(Q, K, V)

# Convert to numpy array
attention_weights_np = attention_weights.detach().numpy()

# Draw a heat map
plt.figure(figsize=(10, 6))
sns.heatmap(attention_weights_np, annot=True, cmap='viridis', xticklabels=words, yticklabels=words)
plt.title('Dot-Product Attention Heatmap')
plt.show()
```



شکل ۵-۳: نقشه حرارتی وزن‌های به دست آمده در روش دوم

در این توجه به کلمات مشابه وزن بیشتری داده شده است. همچنین مشاهده می‌شود که وزن کلمه Raha و رها که مقادل انگلیسی و فارسی یکدیگر هستند، یا کلمات Iran . Tehran که تهران پایتخت ایران است، وزن بالایی داده شده است.

۴ توابع طراحی مکانیزم توجه چندسر با کتابخانهی پایتورچ

برای این قسمت از لایه توجه چندسر به نام "Multihead Attention" در معماری ترنسفورمر استفاده شده است. در ابتدا یک کلاس MultiheadAttention تعریف شده است. این کلاس یک لایه توجه چندسر را با استفاده از شبکه‌های کاملاً متصل برای Q، K و V تعریف می‌کند. همچنین یک لایه خروجی نیز دارد. تابع forward کلاس MultiheadAttention

ورودی‌های Q، K و V را به لایه‌های خود اعمال کرده و سپس محاسبات لایه توجه چندسر را انجام می‌دهد. برای تعریف متغیرها و اجرای مدل MultiheadAttention، تعداد سرها (num_heads) و ابعاد بردارها (hidden_size) تعیین شده و سپس یک نمونه از کلاس MultiheadAttention ایجاد شده است. کد این قسمت در شکل ۱-۴ آورده شده است.

```python
import torch
import torch.nn.functional as F
from torch import nn

# Definition of multihead attention function
Comment Code
class MultiheadAttention(nn.Module):
    def __init__(self, hidden_size, num_heads):
        super(MultiheadAttention, self).__init__()
        self.hidden_size = hidden_size
        self.num_heads = num_heads

        # Define input layers for Q, K and V
        self.linear_q = nn.Linear(hidden_size, hidden_size)
        self.linear_k = nn.Linear(hidden_size, hidden_size)
        self.linear_v = nn.Linear(hidden_size, hidden_size)

        # Definition of output layers
        self.linear_out = nn.Linear(hidden_size, hidden_size)

    def forward(self, Q, K, V):
        # Apply input layers to Q, K and V
        q = self.linear_q(Q)
        k = self.linear_k(K)
        v = self.linear_v(V)

        # Divide vectors Q, K and V into heads
        q = q.view(Q.size(0), -1, self.num_heads, self.hidden_size // self.num_heads).transpose(1, 2)
        k = k.view(K.size(0), -1, self.num_heads, self.hidden_size // self.num_heads).transpose(1, 2)
        v = v.view(V.size(0), -1, self.num_heads, self.hidden_size // self.num_heads).transpose(1, 2)

        # apply attention to each head K and V to the heads
        attention_scores = torch.matmul(q, k.transpose(-1, -2)) / torch.sqrt(torch.tensor(self.hidden_size / self.num_heads, dtype=torch.float32))
        attention_scores = F.softmax(attention_scores, dim=-1)
        output = torch.matmul(attention_scores, v)

        # Combine output from headers
        output = output.transpose(1, 2).contiguous().view(Q.size(0), -1, self.hidden_size)
        output = self.linear_out(output)

        return output

# Number of heads
num_heads = 4
hidden_size = 768
# Definition of the multihead attention model
multihead_attention = MultiheadAttention(hidden_size, num_heads)
```

شکل ۱-۴: کد مکانیزم توجه چندسر با کتابخانهی پایتورچ

برای اعمال MultiheadAttention بر روی بردارهای Q، K و V، با استفاده از نمونه multihead_attention، تابع توجه چندسر بر روی بردارهای Q، K و V اعمال شده و خروجی نهایی چاپ شده است. مقدار توجه چندسر نهایی در شکل ۲-۴ آورده شده است.

```
# apply multihead attention on Q, K and V vectors
output_multihead = multihead_attention(Q, K, V)

# Print the final output
print("Output Tensor (Multihead Attention):")
print(output_multihead)
```

✓ 0.1s

```
Output Tensor (Multihead Attention):
tensor([[[ 0.0505,  0.1375, -0.1923,  ...,  0.0284, -0.0646,  0.1645],
         [-0.0361,  0.1985, -0.0766,  ...,  0.1029, -0.0478,  0.0989],
         [ 0.0442,  0.1211, -0.0496,  ...,  0.1436, -0.0588,  0.0966],
         [ 0.1008,  0.1725, -0.0806,  ...,  0.2228,  0.0059,  0.1327],
         [ 0.0135,  0.1760, -0.1147,  ...,  0.0359, -0.0470,  0.0827],
         [ 0.0379,  0.1396, -0.0891,  ...,  0.1261, -0.0481,  0.1475]],

        [[ 0.1057, -0.2296,  0.0237,  ...,  0.0770, -0.1283, -0.2021],
         [ 0.1016, -0.2727,  0.0512,  ..., -0.0265, -0.1863, -0.2060],
         [ 0.1276, -0.1750, -0.0046,  ..., -0.0302, -0.2084, -0.1426],
         [ 0.0900, -0.2580,  0.0171,  ..., -0.0503, -0.1992, -0.2090],
         [ 0.0758, -0.2706, -0.0060,  ...,  0.0399, -0.1069, -0.1981],
         [ 0.0592, -0.1791,  0.0051,  ...,  0.0515, -0.1761, -0.1770]],

        [[ 0.2070,  0.0944,  0.1567,  ...,  0.0907, -0.1684, -0.0468],
         [ 0.0639,  0.1487,  0.0748,  ...,  0.1574, -0.1920, -0.0785],
         [ 0.1979,  0.1000,  0.1159,  ...,  0.0439, -0.1498, -0.0188],
         [ 0.2237,  0.0962,  0.1334,  ...,  0.0760, -0.2544, -0.1045],
         [ 0.1754,  0.0993,  0.1419,  ...,  0.0887, -0.2212, -0.1205],
         [ 0.2095,  0.1189,  0.0910,  ...,  0.0118, -0.2149, -0.0112]],

        ...,

        [[ 0.1388, -0.1359,  0.0777,  ...,  0.0304,  0.1091,  0.0755],
...
         [-0.0517, -0.0933,  0.0284,  ...,  0.1165, -0.0295, -0.0461],
         [-0.0629, -0.1126, -0.0513,  ...,  0.0744, -0.0871,  0.0162],
         [-0.1294, -0.1103, -0.0712,  ...,  0.1162, -0.0492,  0.0383]]],
       grad_fn=<ViewBackward0>)
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

شکل ۲-۴: مقدار توجه چندسر نهایی و نقشه حرارتی آن

در نهایت یک تابع plot_attention_heatmap تعریف شده است که از آن برای رسم نمودار حرارتی از ماتریس وزن‌های توجه استفاده شده است که در شکل قابل مشاده است.

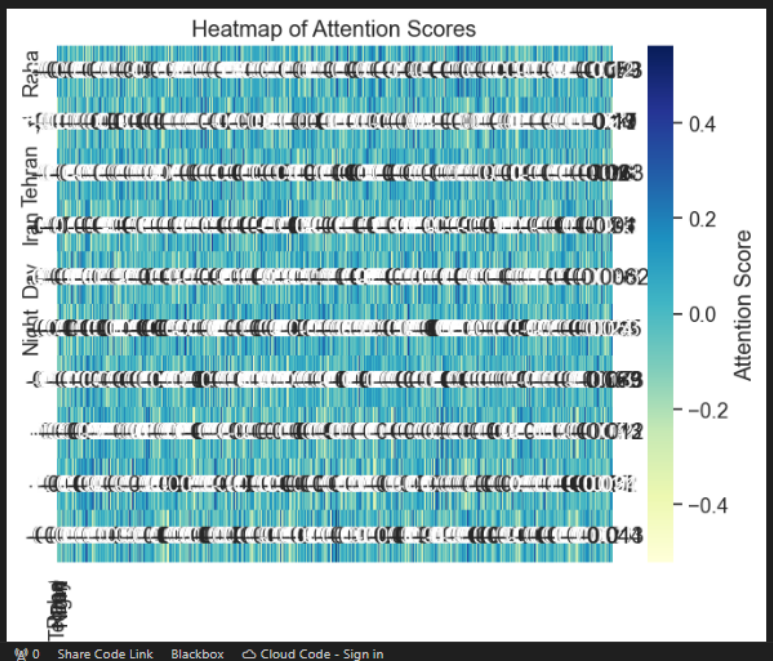
```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Function to draw heatmap attention
Comment Code
def plot_attention_heatmap(attention_matrix, words):
    sns.set(font_scale=1.2)
    plt.figure(figsize=(8, 6))
    ax = sns.heatmap(attention_matrix, annot=True, xticklabels=words, yticklabels=words, cmap="YlGnBu", cbar_kws={'label': 'Attention Score'})
    plt.title('Heatmap of Attention Scores')
    plt.show()

# Averaging over heads
attention_matrix_avg = np.mean(attention_matrix_multihead, axis=1).squeeze()

# Draw heatmap attention
plot_attention_heatmap(attention_matrix_avg, words)
```

✓ 31.3s



18 ⚠ 0 Share Code Link Blackbox Cloud Code - Sign in

شکل ۳-۴: نمودار حرارتی توجه چند سر

نمودار توجه محاسبه شده به هم ریخته می‌باشد. برای توجه چند سر یک کد دیگر نیز زده شده است.

برای این کار ابتدا یک کلاس به نام MultiHeadAttention تعریف شده است. این لایه توجه چندسر با استفاده از چند سر (head) مختلف از بردارهای Query (Q)، Key (K) و Value (V) عمل می‌کند. سپست تابع forward، Q، K و V را از طریق لایه‌های خطی مختلف تبدیل می‌کند و سپس توجه چندسر را محاسبه می‌کند. کد این قسمت در شکل ۴-۴ آورده شده است.

```python
import torch.nn.functional as F

Comment Code
class MultiHeadAttention(torch.nn.Module):
    def __init__(self, embedding_dim, num_heads):
        super(MultiHeadAttention, self).__init__()
        self.embedding_dim = embedding_dim
        self.num_heads = num_heads
        self.head_dim = embedding_dim // num_heads

        assert (
            self.head_dim * num_heads == embedding_dim
        ), "Embedding dimension needs to be divisible by the number of heads"

        self.W_q = torch.nn.Linear(embedding_dim, embedding_dim, bias=False)
        self.W_k = torch.nn.Linear(embedding_dim, embedding_dim, bias=False)
        self.W_v = torch.nn.Linear(embedding_dim, embedding_dim, bias=False)
        self.W_o = torch.nn.Linear(embedding_dim, embedding_dim)

    def forward(self, Q, K, V):
        batch_size, seq_length, _ = Q.size()

        # Linear transformations for Q, K, and V
        Q = self.W_q(Q).view(batch_size, seq_length, self.num_heads, self.head_dim).transpose(1, 2)
        K = self.W_k(K).view(batch_size, seq_length, self.num_heads, self.head_dim).transpose(1, 2)
        V = self.W_v(V).view(batch_size, seq_length, self.num_heads, self.head_dim).transpose(1, 2)

        # Attention scores and scaling
        attention_scores = torch.matmul(Q, K.transpose(-2, -1)) / torch.sqrt(torch.tensor(self.head_dim, dtype=torch.float32))

        # Softmax to get attention weights
        attention_weights = F.softmax(attention_scores, dim=-1)

        # Apply attention weights to V
        attention_output = torch.matmul(attention_weights, V)

        # Reshape and linear transformation for the output
        attention_output = attention_output.transpose(1, 2).contiguous().view(batch_size, seq_length, self.embedding_dim)
        output = self.W_o(attention_output)

        return output, attention_weights
```

شکل ۴-۴: کد محاسبه توجه چندسر به روش دوم

برای تعریف و اجرای مدل MultiHeadAttention، تعداد سرها (num_heads)، ابعاد بردارها (embedding_dim) و سایز دسته (batch_size) تعیین شده و سپس یک نمونه از کلاس MultiHeadAttention ایجاد شده است. که کد آن در شکل ۵-۴ آورده شده است.

```
# Number of heads
num_heads = 8

# Create Q, K and V vectors
batch_size = 10
seq_length = len(words)
embedding_dim = 768

Q = torch.randn(batch_size, seq_length, embedding_dim)
K = torch.randn(batch_size, seq_length, embedding_dim)
V = torch.randn(batch_size, seq_length, embedding_dim)

# Create multi-head attention layer
multihead_attention = MultiHeadAttention(embedding_dim, num_heads)

# Calculation of multi-headed attention
attention_output, attention_weights = multihead_attention(Q, K, V)
```

شکل ۵-۴: کد تعریف و اجرای مدل *MultiHeadAttention* به روش دوم

در مرحله بعد با استفاده از نمونه multihead_attention، تابع توجه چندسر بر روی بردارهای Q، K و V اعمال شده و خروجی نهایی چاپ شده است که آن را می‌توان در شکل ۶-۴ مشاهده نمود.

```
# Print the output of the note
print("Attention Output:")
print(attention_output)
```

```
Attention Output:
tensor([[[-0.0352, -0.1764, -0.1958,  ..., -0.0218, -0.1767,  0.0068],
         [-0.1270, -0.2041, -0.2472,  ..., -0.0748, -0.1786,  0.0749],
         [-0.0706, -0.1551, -0.2057,  ..., -0.0293, -0.1509,  0.0867],
         [-0.0336, -0.1780, -0.2422,  ..., -0.0283, -0.1320,  0.0495],
         [-0.1132, -0.2056, -0.2034,  ..., -0.0813, -0.1530,  0.1008],
         [-0.0887, -0.1434, -0.2315,  ..., -0.0271, -0.1077,  0.1103]],

        [[ 0.0164,  0.1018, -0.0757,  ...,  0.0735,  0.0428,  0.0386],
         [ 0.0685,  0.0282, -0.0185,  ...,  0.0140, -0.0226,  0.0614],
         [-0.0053,  0.0735, -0.0795,  ...,  0.0593,  0.0742,  0.0012],
         [ 0.1464,  0.0916, -0.0501,  ...,  0.0766,  0.0917, -0.0259],
         [ 0.0498, -0.0133, -0.0746,  ...,  0.0512, -0.0534,  0.0627],
         [ 0.0217, -0.0023, -0.0651,  ...,  0.1096,  0.0582,  0.0585]],

        [[-0.0102,  0.1594,  0.1514,  ...,  0.2154, -0.0727,  0.0443],
         [ 0.0652,  0.1807,  0.1115,  ...,  0.3256, -0.1004,  0.1361],
         [-0.0344,  0.1659,  0.0898,  ...,  0.2011, -0.0893,  0.0257],
         [-0.0022,  0.1522,  0.0804,  ...,  0.2501, -0.1311,  0.0661],
         [ 0.0112,  0.1158,  0.0168,  ...,  0.2006, -0.1204,  0.0681],
         [-0.0180,  0.1654,  0.1502,  ...,  0.2190, -0.0644,  0.0581]],

        ...,

        [[ 0.1031, -0.2854, -0.0759,  ...,  0.2264, -0.1410,  0.2016],
...
         [ 0.0744, -0.0744,  0.1503,  ..., -0.0525, -0.0935,  0.1267],
         [ 0.0903, -0.0906,  0.1249,  ..., -0.0157, -0.1188,  0.1582],
         [ 0.1399, -0.1003,  0.0478,  ..., -0.0024, -0.1490,  0.0907]]],
       grad_fn=<ViewBackward0>)
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

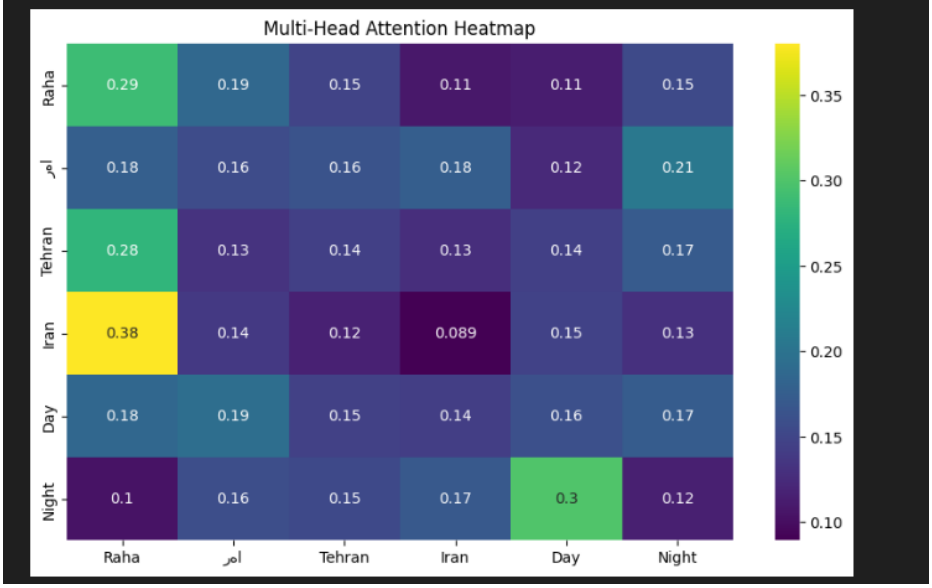
شکل ۶-۴: اعمال تابع توجه چندسر بر روی بردارهای Q، K و V به روش دوم

در نهایت ماتریس وزن‌های توجه از بردارهای attention_matrix_multihead با گرفتن میانگین بر روی سرها محاسبه شده و نمودار حرارتی توجه رسم شده است. نمودار حرارتی به دست آمده از این توجه را می‌توان در شکل ۴-۷ مشاهده نمود.

```python
import matplotlib.pyplot as plt
import seaborn as sns

# Convert to numpy array
attention_weights_np = attention_weights.detach().numpy()

# Draw a heat map
plt.figure(figsize=(10, 6))
sns.heatmap(attention_weights_np[0][0], annot=True, cmap='viridis', xticklabels=words, yticklabels=words)
plt.title('Multi-Head Attention Heatmap')
plt.show()
```



شکل ۷-۴: نمودار حرارتی به دست آمده از توجه به روش دوم

در نمودار حرارتی توجه محاسبه شده مشاهده می‌شود که وزن کلمات یکسان همچنان زیاد است اما همچنان بیشتر از وزن کلمات معادل یا نزدیک به یکدیگر هستند. برای مثال وزن کلمه Raha و Raha نسبت به وزن کلمه Raha و رها بیشتر است. همچنین وزن کلمه Day و Night زیاد است. این توجه نسبت به توجه‌های دیگر وزن‌دهی بهتری داشته است که این مسئله به دلیل توجه چندگانه (چندسر) مدل به کلمات است.

۵ منابع:

https://pytorch.org/docs/stable/nn.html