Below is a concise yet friendly explanation of the attention mechanism for assignment introduction:

Attention Mechanism (Adapted from "Attention Is All You Need")

The attention mechanism, introduced in Attention Is All You Need (Vaswani et al., 2017), processes inputs represented as vectors (each row is a token embedding of dimension D). We compute three sets of vectors: queries (Q), keys (K), and values (V). The core steps are:

1. **Linear Transformations:**

Let X be the input matrix, where each of the rows corresponds to a token in the input sequence, and each row is a d-dimensional embedding vector.

To compute attention, we first project X into three different representations using learned weight matrices:

Each input vector is transformed into Q, K, and V using learnable weights.

$$
\begin{aligned}
Q_i &= XW_i^Q, \\
K_i &= XW_i^K, \\
V_i &= XW_i^V.
\end{aligned}
$$

Each head (i) has its own learnable parameters W_i^Q, W_i^K, and W_i^V, which transform the input into queries, keys, and values, respectively.

2. **Scaled Dot-Product Attention:**

$$
\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V. \tag{1}
$$

Here, QK^T produces a matrix of scores that measures how relevant each "query" position is to every "key" position.

The softmax function converts these scores into attention weights (non-negative values that sum to 1 across each row).

These weights are then used to combine the values *V* to produce the final output.

3. **Multi-Head Attention:**

$$
\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)
$$

$$
\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V).
$$

Multiple attention heads allow the model to attend to different aspects of the input simultaneously. Their outputs are concatenated and linearly transformed to produce the final result.

Note: In this assignment, you are only required to experiment with the provided Q, K, and V matrices to perform the matrix multiplication

Self-attention Computer Assignment

Implement the multi-head self-attention operation, taking in a set of N vectors of D dimensions and outputting a matrix of the same size. Do this without relying on neural network libraries, but rather write directly the required operations in NumPy.

In [12]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [13]:
```
# Data size
N = 5
D = 6

X = [[ 0.7, -0.8, -1.2,  -1.,  -0., -0.3],
     [ 2.7,  0.1,  1.6,  1.8,  1.5,  0.3],
     [ 0.1,  2.6, -0.1, -1.3, -0.5, -0.7],
     [ 1.1,  1.5,   1., -0.5,  0.4,  0.4],
     [-0.7, -0.7,  0.7, -1.5, -0.8,  1. ]]

Wq = [[-1.7,  1.6,  0.9, -0.5,  0.4,  -1.],
      [-0.4,  1. , -0.3,  1. ,  0.5,  1.1],
      [ 0.4, -0.9,  -1.,  0.5, -1.4,  0. ],
      [ 0.3,  1.4, -1.2,  0.2,  0.1,  1.6],
      [-0.8,  0.8, -0.7, -1.3,  0.3,  0.8],
      [ 1.1,  0.3, -1.5, -2.3,  2.2, -0.7]]

Wk = [[ 0.3, -0.4, -1.3,  0.3, -1.7,  1.1],
      [-2.3, -1.1,  0.6, -1.2,  2.2,  0.3],
      [ 1.1, -0.4, -0.5,  1.9, -1.1, -1.2],
      [-0.4,  1. , -1.7,  0. , -3.3, -1.4],
      [-0.9, -1.1, -1. ,  1.4,  1.3,  1.2],
      [-0.7,  0.4,  0.4, -1.4, -0.2, -0.5]]

Wv = [[-0.1,  0.7,  1. , -0.1,  1.6,  0.9],
      [ 0.4, -1. , -0.7, -0.6, -0.9, -0.1],
      [-0.4,  0.5, -1.4,  0.1,  0.6,  0.4],
      [ 1.4, -1.3, -1.3, -0.6,  1.6, -0.2],
      [-0.4, -0.6, -1.4, -1. ,  0.4, -0.8],
      [ 0.2,  0.5,  0.4, -0.5,  1.4,  2.3]]



X = np.array(X)
Wq = np.array(Wq)
Wk = np.array(Wk)
Wv = np.array(Wv)
```

(a) Implement the self-attention operation

In [14]:
```
def self_attention(X, Wq, Wk, Wv):
    Q = X @ Wq
    K = X @ Wk
```

```
    V = X @ Wv

    d_k = K.shape[1]
    scores = Q @ K.T
    scores /= np.sqrt(d_k)
    exp_scores = np.exp(scores)
    attention_weights = exp_scores / np.sum(exp_scores, axis=1, keepdims=True)
    output = attention_weights @ V

    return output, attention_weights
```

In [15]:
```
# Compute the output
output, attention_weights = self_attention(X, Wq, Wk, Wv)

# Print in a nice format
np.set_printoptions(precision=1)
print("Self-Attention Output:\n", output)
print("Self-Attention Matrix:\n", attention_weights)
```

```
Self-Attention Output:
 [[-0.7 -0.9  0.5  0.1 -5.5 -1.1]
 [-0.4  0.6  0.6 -0.6  2.   0.5]
 [ 0.3 -0.1 -2.4 -1.9  4.9  1.8]
 [-0.7 -0.7  0.4 -0.1 -4.5 -0.7]
 [-2.   3.3  2.1  1.6 -1.3  2.8]]
Self-Attention Matrix:
 [[4.9e-07 4.0e-14 9.9e-01 1.7e-05 6.1e-03]
 [4.8e-01 3.6e-01 1.5e-01 2.1e-02 4.0e-04]
 [1.9e-02 5.8e-01 9.4e-02 2.8e-01 3.2e-02]
 [3.0e-02 1.2e-02 8.5e-01 9.9e-02 1.1e-02]
 [4.7e-04 7.3e-03 1.6e-02 4.0e-02 9.4e-01]]
```

(b) Implement multi-head attention, using the previously implemented function

In [16]:
```
def multi_head_attention(X, Wq, Wk, Wv, H):
    # split the dimension D into H heads
    head_idx = np.split(np.arange(Wk.shape[1]), H)
    head_outputs = []
    head_weights = []
    for idx in head_idx:
        output, weights = self_attention(X, Wq[:, idx], Wk[:, idx], Wv[:, idx])
        head_outputs.append(output)
        head_weights.append(weights)
    output = np.concatenate(head_outputs, axis=1)

    return output, head_weights
```

In [17]:
```
# Compute multi-head attention
H = 3
attention_output = multi_head_attention(X, Wq, Wk, Wv, H)
# Again print the requested results
print("Self-Attention Output:\n", attention_output[0])
print("Self-Attention Weights:")
for i, m in enumerate(attention_output[1]):
    print(f"Head {i+1}\n {m}\n")
```

```
Self-Attention Output:
 [[-0.7 -0.9  0.7  0.2 -1.5  2.9]
 [-1.1  0.9 -3.9 -2.9 -3.  -0.5]
 [-0.7 -0.9  1.6  1.2  5.8  1.5]
 [-0.7 -0.7 -3.8 -2.8 -5.1 -0.9]
 [-0.2  0.3  1.   0.2 -1.3  3. ]]
Self-Attention Weights:
Head 1
 [[1.8e-04 1.2e-03 9.5e-01 4.6e-02 2.3e-05]
 [5.1e-01 1.7e-02 3.6e-01 1.0e-02 1.0e-01]
 [6.4e-04 2.5e-03 9.4e-01 5.4e-02 9.7e-05]
 [3.8e-02 2.6e-02 8.4e-01 8.8e-02 1.0e-02]
 [2.6e-02 3.2e-01 4.5e-02 5.3e-01 7.7e-02]]

Head 2
 [[3.1e-05 2.0e-18 9.0e-01 4.8e-07 9.7e-02]
 [6.1e-04 1.0e+00 9.6e-05 8.1e-04 9.0e-07]
 [3.2e-03 1.1e-01 3.8e-04 1.5e-02 8.7e-01]
 [1.0e-02 9.7e-01 2.3e-03 1.8e-02 2.0e-03]
 [1.1e-01 7.5e-04 7.9e-01 4.0e-02 5.4e-02]]

Head 3
 [[3.1e-05 3.9e-05 1.8e-02 3.6e-03 9.8e-01]
 [4.3e-01 4.7e-03 4.9e-01 7.1e-02 6.8e-03]
 [2.3e-01 6.7e-01 2.2e-02 6.3e-02 1.5e-02]
 [1.4e-02 4.5e-05 9.1e-01 2.7e-02 4.4e-02]
 [2.8e-06 3.9e-03 1.0e-04 8.4e-04 1.0e+00]]
```

(c+d) Provide the answers/explanations requested in the problem sheet:

1. Why the results are different?
2. What happens if you change the order of two inputs=

For $H = 1$ we have

```
Self-Attention Output:
 [[-0.7 -0.9  0.5  0.1 -5.5 -1.1]
 [-0.4  0.6  0.6 -0.6  2.   0.5]
 [ 0.3 -0.1 -2.4 -1.9  4.9  1.8]
 [-0.7 -0.7  0.4 -0.1 -4.5 -0.7]
 [-2.   3.3  2.1  1.6 -1.3  2.8]]
Self-Attention Matrix:
 [[4.9e-07 4.0e-14 9.9e-01 1.7e-05 6.1e-03]
 [4.8e-01 3.6e-01 1.5e-01 2.1e-02 4.0e-04]
 [1.9e-02 5.8e-01 9.4e-02 2.8e-01 3.2e-02]
 [3.0e-02 1.2e-02 8.5e-01 9.9e-02 1.1e-02]
 [4.7e-04 7.3e-03 1.6e-02 4.0e-02 9.4e-01]]
```

For $H = 3$ we have

```
Self-Attention Output:
 [[-0.7 -0.9  0.7  0.2 -1.5  2.9]
 [-1.1  0.9 -3.9 -2.9 -3.  -0.5]
```

```
 [-0.7 -0.9  1.6  1.2  5.8  1.5]
 [-0.7 -0.7 -3.8 -2.8 -5.1 -0.9]
 [-0.2  0.3  1.   0.2 -1.3  3. ]]
Self-Attention Weights:
Head 1
 [[1.8e-04 1.2e-03 9.5e-01 4.6e-02 2.3e-05]
 [5.1e-01 1.7e-02 3.6e-01 1.0e-02 1.0e-01]
 [6.4e-04 2.5e-03 9.4e-01 5.4e-02 9.7e-05]
 [3.8e-02 2.6e-02 8.4e-01 8.8e-02 1.0e-02]
 [2.6e-02 3.2e-01 4.5e-02 5.3e-01 7.7e-02]]

Head 2
 [[3.1e-05 2.0e-18 9.0e-01 4.8e-07 9.7e-02]
 [6.1e-04 1.0e+00 9.6e-05 8.1e-04 9.0e-07]
 [3.2e-03 1.1e-01 3.8e-04 1.5e-02 8.7e-01]
 [1.0e-02 9.7e-01 2.3e-03 1.8e-02 2.0e-03]
 [1.1e-01 7.5e-04 7.9e-01 4.0e-02 5.4e-02]]

Head 3
 [[3.1e-05 3.9e-05 1.8e-02 3.6e-03 9.8e-01]
 [4.3e-01 4.7e-03 4.9e-01 7.1e-02 6.8e-03]
 [2.3e-01 6.7e-01 2.2e-02 6.3e-02 1.5e-02]
 [1.4e-02 4.5e-05 9.1e-01 2.7e-02 4.4e-02]
 [2.8e-06 3.9e-03 1.0e-04 8.4e-04 1.0e+00]]
```

The results are different because the shape of the attention matrix is $N \times N$, where N is the number of inputs, i.e. it is independent of the dimension D of the input. In other words, if we compute the self-attention with H heads, we get H times $N \times N$ attention matrices, instead of just a single attention matrix when using a single head. So the attention matrix is not "sliced" even though the other other matrices (Q, K, V) are, leading to a different result because the values V are *routed* differently to create the output in the two cases. This leads to the multi-head self-attention being more expressive, since each head can pay attention to different characteristics of the input simultaneously.

In [21]:
```
# change the order of first two inputs
print("X\n",X)
X = X[[1,0,2,3,4],:]
print("X after permutation\n", X)
```

```
X
 [[ 0.7 -0.8 -1.2 -1.  -0.  -0.3]
 [ 2.7  0.1  1.6  1.8  1.5  0.3]
 [ 0.1  2.6 -0.1 -1.3 -0.5 -0.7]
 [ 1.1  1.5  1.  -0.5  0.4  0.4]
 [-0.7 -0.7  0.7 -1.5 -0.8  1. ]]
X after permutation
 [[ 2.7  0.1  1.6  1.8  1.5  0.3]
 [ 0.7 -0.8 -1.2 -1.  -0.  -0.3]
 [ 0.1  2.6 -0.1 -1.3 -0.5 -0.7]
 [ 1.1  1.5  1.  -0.5  0.4  0.4]
 [-0.7 -0.7  0.7 -1.5 -0.8  1. ]]
```

In [22]:
```
H = 1
attention_output = multi_head_attention(X, Wq, Wk, Wv, H)
```

```
# Again print the requested results
print("Self-Attention Output:\n", attention_output[0])
print("Self-Attention Weights:")
for i, m in enumerate(attention_output[1]):
    print(f"Head {i+1}\n {m}\n")
```

```
Self-Attention Output:
 [[-0.4  0.6  0.6 -0.6  2.   0.5]
 [-0.7 -0.9  0.5  0.1 -5.5 -1.1]
 [ 0.3 -0.1 -2.4 -1.9  4.9  1.8]
 [-0.7 -0.7  0.4 -0.1 -4.5 -0.7]
 [-2.   3.3  2.1  1.6 -1.3  2.8]]
Self-Attention Weights:
Head 1
 [[3.6e-01 4.8e-01 1.5e-01 2.1e-02 4.0e-04]
 [4.0e-14 4.9e-07 9.9e-01 1.7e-05 6.1e-03]
 [5.8e-01 1.9e-02 9.4e-02 2.8e-01 3.2e-02]
 [1.2e-02 3.0e-02 8.5e-01 9.9e-02 1.1e-02]
 [7.3e-03 4.7e-04 1.6e-02 4.0e-02 9.4e-01]]
```

In [23]:
```
H = 3
attention_output = multi_head_attention(X, Wq, Wk, Wv, H)
# Again print the requested results
print("Self-Attention Output:\n", attention_output[0])
print("Self-Attention Weights:")
for i, m in enumerate(attention_output[1]):
    print(f"Head {i+1}\n {m}\n")
```

```
Self-Attention Output:
 [[-1.1  0.9 -3.9 -2.9 -3.  -0.5]
 [-0.7 -0.9  0.7  0.2 -1.5  2.9]
 [-0.7 -0.9  1.6  1.2  5.8  1.5]
 [-0.7 -0.7 -3.8 -2.8 -5.1 -0.9]
 [-0.2  0.3  1.   0.2 -1.3  3. ]]
Self-Attention Weights:
Head 1
 [[1.7e-02 5.1e-01 3.6e-01 1.0e-02 1.0e-01]
 [1.2e-03 1.8e-04 9.5e-01 4.6e-02 2.3e-05]
 [2.5e-03 6.4e-04 9.4e-01 5.4e-02 9.7e-05]
 [2.6e-02 3.8e-02 8.4e-01 8.8e-02 1.0e-02]
 [3.2e-01 2.6e-02 4.5e-02 5.3e-01 7.7e-02]]

Head 2
 [[1.0e+00 6.1e-04 9.6e-05 8.1e-04 9.0e-07]
 [2.0e-18 3.1e-05 9.0e-01 4.8e-07 9.7e-02]
 [1.1e-01 3.2e-03 3.8e-04 1.5e-02 8.7e-01]
 [9.7e-01 1.0e-02 2.3e-03 1.8e-02 2.0e-03]
 [7.5e-04 1.1e-01 7.9e-01 4.0e-02 5.4e-02]]

Head 3
 [[4.7e-03 4.3e-01 4.9e-01 7.1e-02 6.8e-03]
 [3.9e-05 3.1e-05 1.8e-02 3.6e-03 9.8e-01]
 [6.7e-01 2.3e-01 2.2e-02 6.3e-02 1.5e-02]
 [4.5e-05 1.4e-02 9.1e-01 2.7e-02 4.4e-02]
 [3.9e-03 2.8e-06 1.0e-04 8.4e-04 1.0e+00]]
```

The output of the self-attention is thus *equivariant* to permutations of the input X. However, the attention weights need to be different for this to be true. It takes as input a

set, paying no attention to ordering. The ordering can be taken into account by positional encoding.