8. Self-Attention Mechanism

In [ ]:
```
# Import PyTorch for tensor operations and neural network functions
import torch
import torch.nn.functional as F
```

In [ ]:
```
# Create a random input tensor: (batch=1, sequence_length=3, embedding_dim=4)
x = torch.rand(1, 3, 4)
# In self-attention, use the same tensor for queries (Q), keys (K), and values (V)
Q, K, V = x, x, x
```

In [ ]:
```
# Compute attention scores by dot product of Q and K, normalized by sqrt(embedding_dim)
scores = torch.matmul(Q, K.transpose(-2, -1)) / (4 ** 0.5)
# Apply softmax to get attention weights (probabilities)
weights = F.softmax(scores, dim=-1)
# Compute the output as the weighted sum of values (V)
output = torch.matmul(weights, V)
```

In [4]:
```
# Print the attention weights (how much each token attends to others)
print("Attention Weights:", weights)

# Print the output tensor (contextualized representations)
print("Output:", output)
```

```
Attention Weights: tensor([[[0.4601, 0.2769, 0.2629],
         [0.3874, 0.3236, 0.2890],
         [0.3707, 0.2913, 0.3381]]])
Output: tensor([[[0.6630, 0.4578, 0.5754, 0.5707],
         [0.6336, 0.4323, 0.5330, 0.5781],
         [0.6498, 0.4231, 0.5034, 0.5734]]])
```

In [ ]: