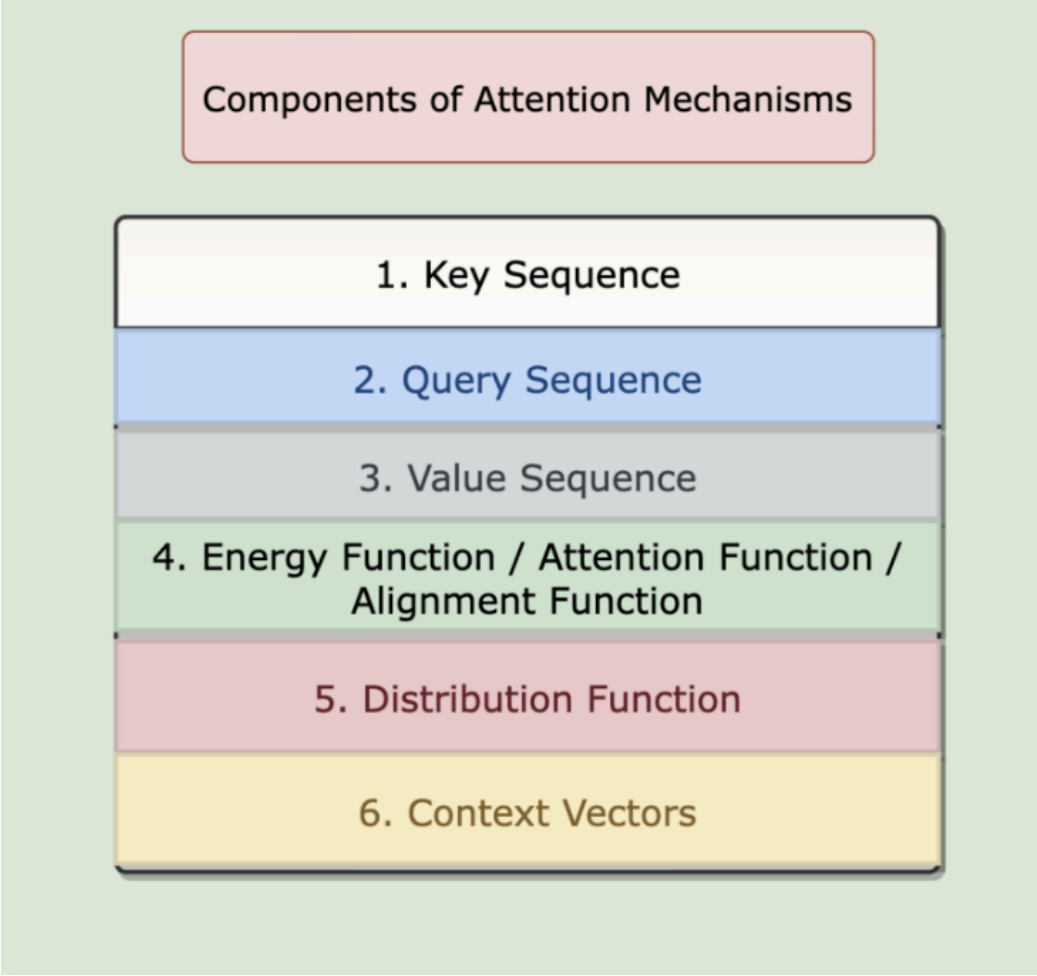
Lecture 9: Attention

What is Attention?

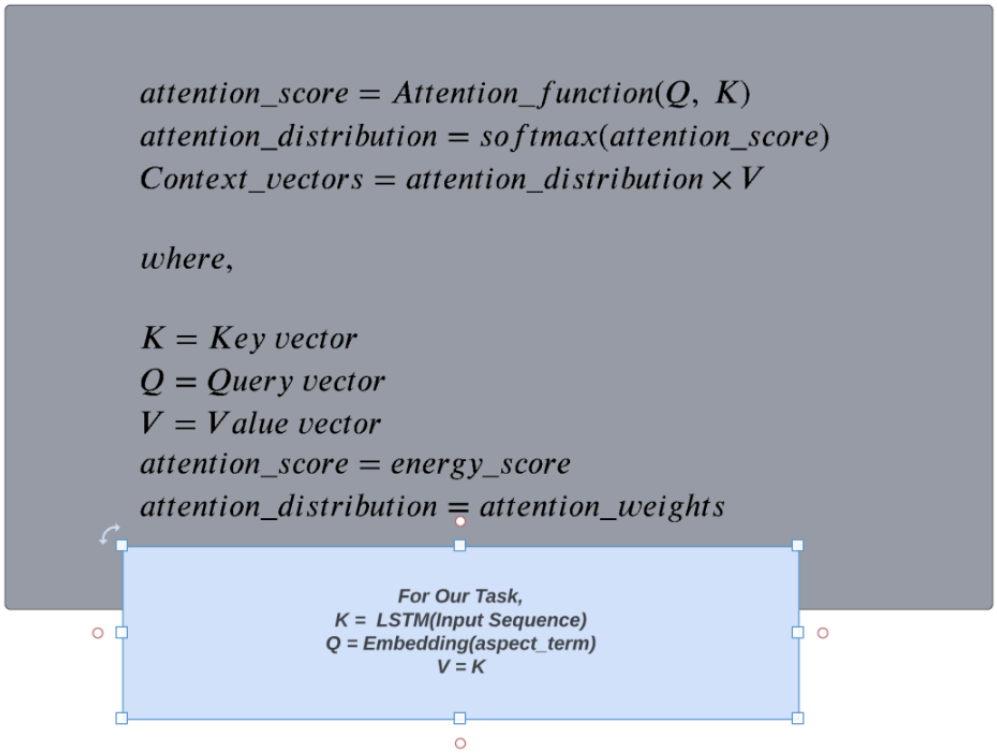
Attention mechanisms Consist of the following Components:

Attention is proposed as a solution to the limitation of the Encoder-Decoder model which encodes the input sequence to one fixed-length vector from which to decode the output at each time step.

1. `Key(K)`
2. `Query(Q)`
3. `Value(V)`
4. `Energy Function(a)`
5. `Alignment distribution(g)`
6. `Final Representation(C)`



Formula for Attention



1. Key

K encodes the data features whereupon attention is computed.

2. Query

The attention mechanism will emphasize the input elements relevant to the task according to Q.Q is represented by embeddings of actual textual queries, contextual information, etc.

3. Attention Function/ Energy function

From the Keys and Query, a vector E of n energy scores/attention score e_i is computed through an alignment function α

$E=\alpha(Q,K)$

4. Alignment Distribution

Attention scores are then transformed into attention weights using distribution function g

$W=g(E)$

Such weights are the outcome of the core attention mechanism. The commonest distribution function is the softmax.

5. Value

Many tasks require the computation of new representations of the keys. In such cases, it is common to have another input element; a sequence V of n vectors vi , the values, representing the data whereupon the attention computed from K and Q is to be applied.

6. Final Representation.

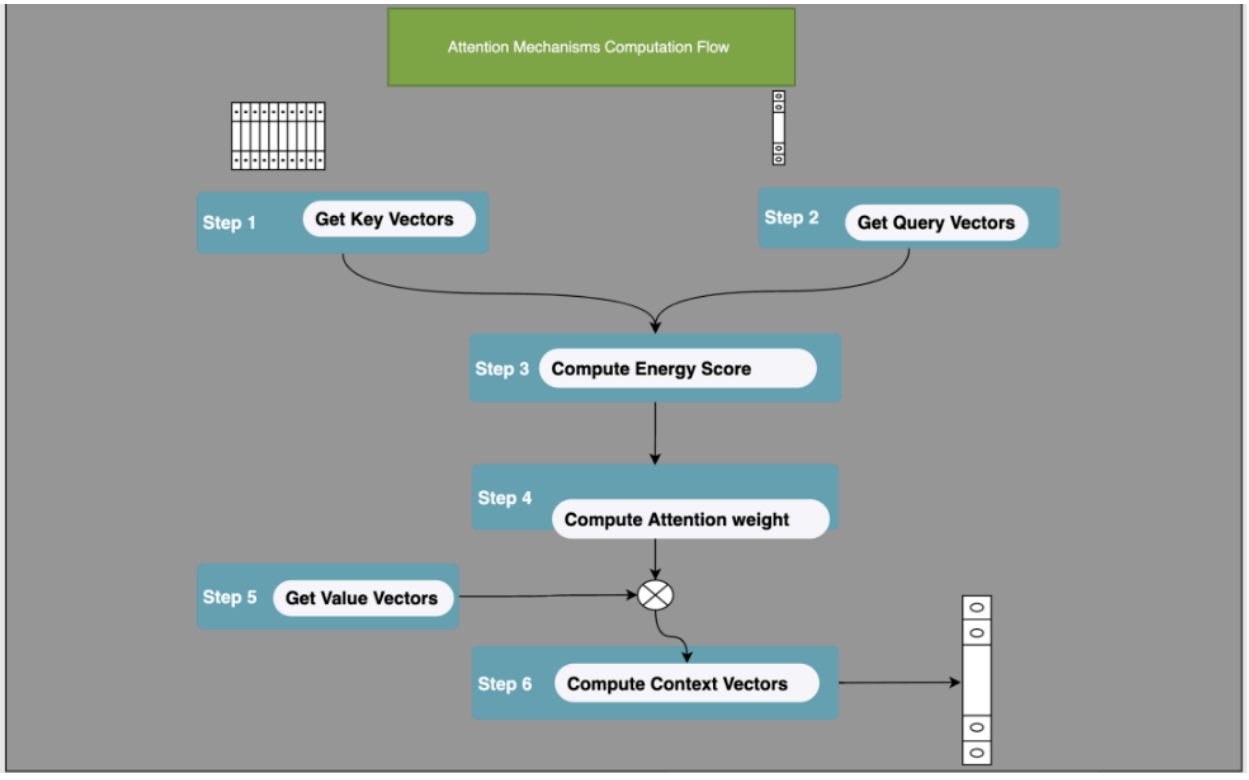
Z=W*V

Each element of V corresponds to one and only one element of K, and the two can be seen as different representations of the same data.

zi=wi*vi,

C=∑dki=1(zi)

How to compute attention?



1. Get KEY

```
#Input
sentences = ['food', 'is', 'delicious', 'but', 'price', 'is', 'high']
```

```
#Dimensionality reduction for easy visualization.
pca = PCA(n_components=5)

# Key sequence
KEY = [glove_vectors[i].tolist() for i in sentences]
KEY = pca.fit_transform(KEY)
```

2. Get Query

```
#Query text
target = [ 'cost']

#Query embedding and dimensionality reduction
QUERY = [glove_vectors[i].tolist() for i in target]
QUERY = pca.transform(QUERY)
```

3. Compute Energy Function

Here, Energy Function is DotProduct.

```
def compute_energy_function(vect, mat):
  # calculate energy/attention fuction
  return np.matmul( mat, np.transpose(vect),)

ENERGY_SCORE = compute_energy_function(QUERY, KEY)
ENERGY_SCORE
```

4. Compute Attention weights

```
def softmax(x):
    # Compute attention weights
    x = np.array(x, dtype=np.float64)
    e_x = np.exp(x)
    return e_x / e_x.sum(axis=0)

ATTN_WEIGHTS = softmax(ENERGY_SCORE)

attn_df = pd.DataFrame(ATTN_WEIGHTS, index = sentences, columns = target)
```

5. VALUE

```
VALUE = np.matrix(KEY)
VALUE[:,1].reshape(1,-1).tolist()
ATTN_WEIGHTS[:, 0][0]
```

6. Context Vectors

```
def apply_attention_scores(attention_weights, annotations):
 # Multiple the annotations by their attention weights
 return  [  annotations[:, i] * attention_weights[i] for i in
range(len(attention_weights))]

applied_attention_1 = apply_attention_scores(ATTN_WEIGHTS,
```

```
np.transpose(VALUE))
def calculate_attention_vector(applied_attention):
 return np.sum(applied_attention, axis=0)

#applied_attention = np.array([[applied_attention_1]])
attention_vector = calculate_attention_vector(applied_attention_1)
```

Attention Mechanisms can be broadly categorized into four categories:

1. Number of Sequences(Self Attention):
    - Distinctive Attention Mechanisms
    - Self Attention Attention Mechanisms
2. Number of Abstraction(Multi-Level Attention)
    - Single Level Attention Mechanisms
    - Multi Level Attention Mechanisms
3. Number of Position(Local Attention)
    - Soft Attention Mechanisms
    - Hard Attention Mechanisms
4. Number of Representation(Multi-head Attention)
    - Single Head Attention Mechanisms
    - Multi Head Attention Mechanisms