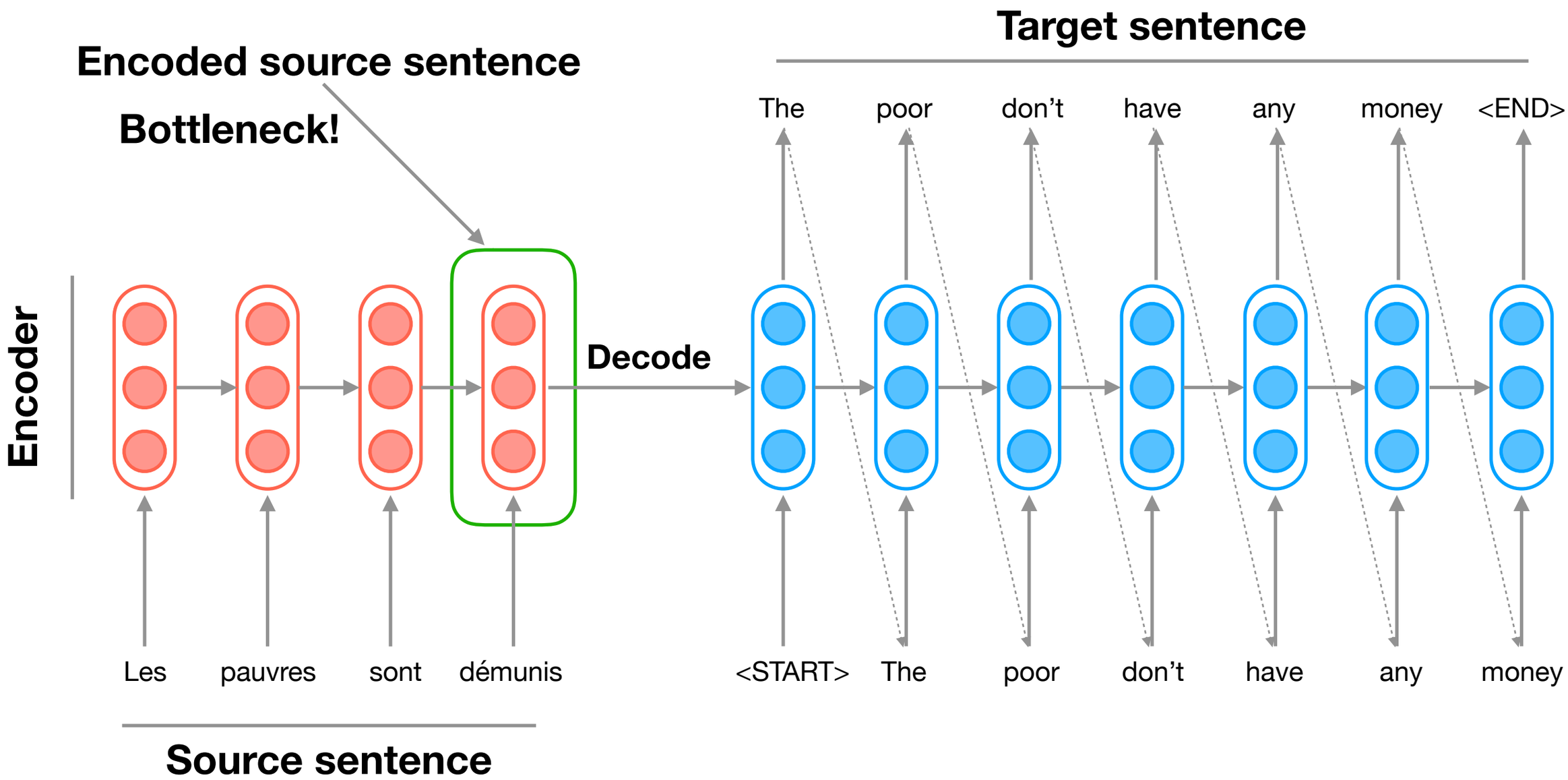
MT

Based on DL MS Course of Boris Zubarev @bobazooba

Sequence to sequence

Inference



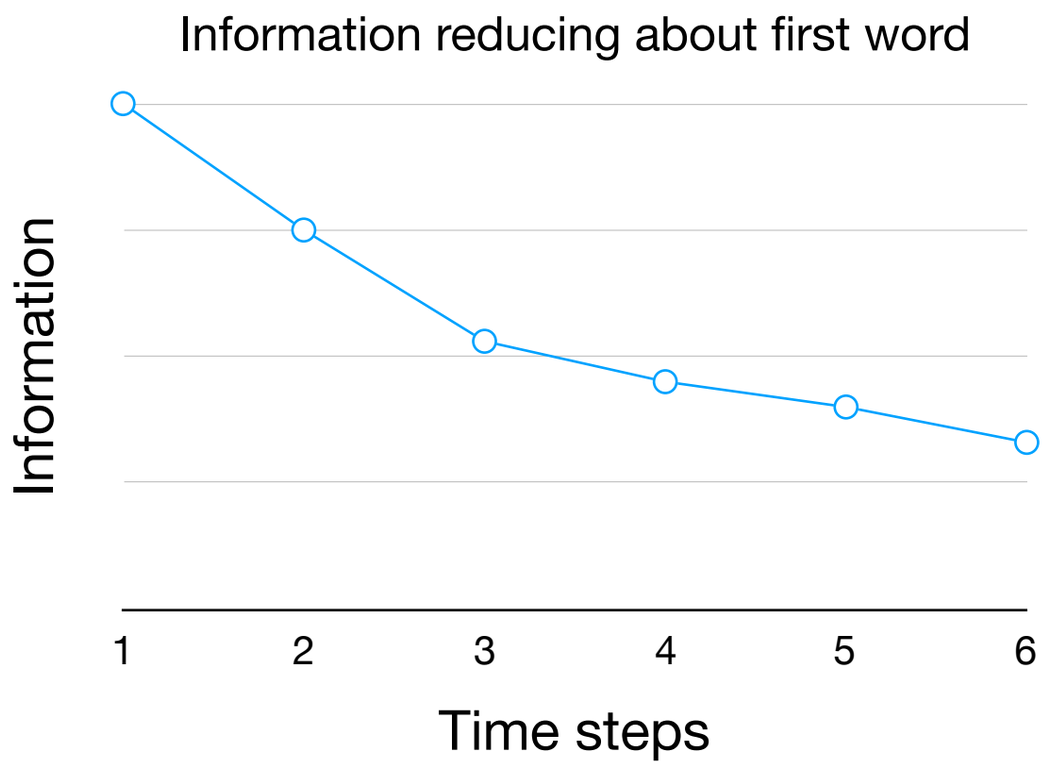
Recurrent Neural Network

+

- Words relationship
- Variable length

—

- Forgot information
- Recurrent dependence



Dot product

- Similarity measure between two vectors

- Cosine similarity (just divide by norms) become measure between 0 and 1

Dot product

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}},$$

```
x1 = np.random.rand(300)
x2 = np.random.rand(300)
```

```
np.dot(x1, x2)
```

```
69.77192646564589
```

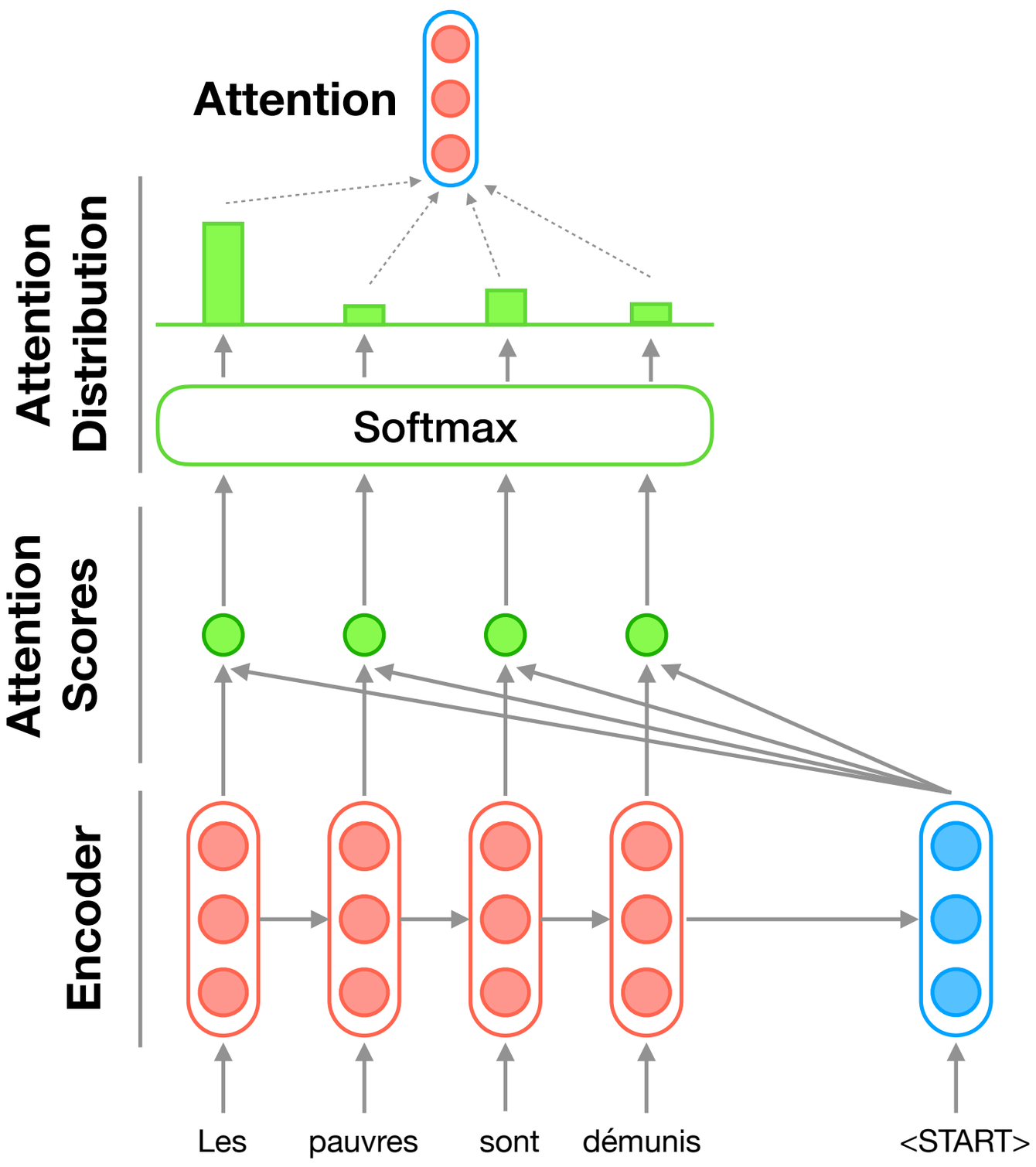
```
np.dot(x1, x2) / (np.linalg.norm(x1) * np.linalg.norm(x2))
```

```
0.7349837217006946
```

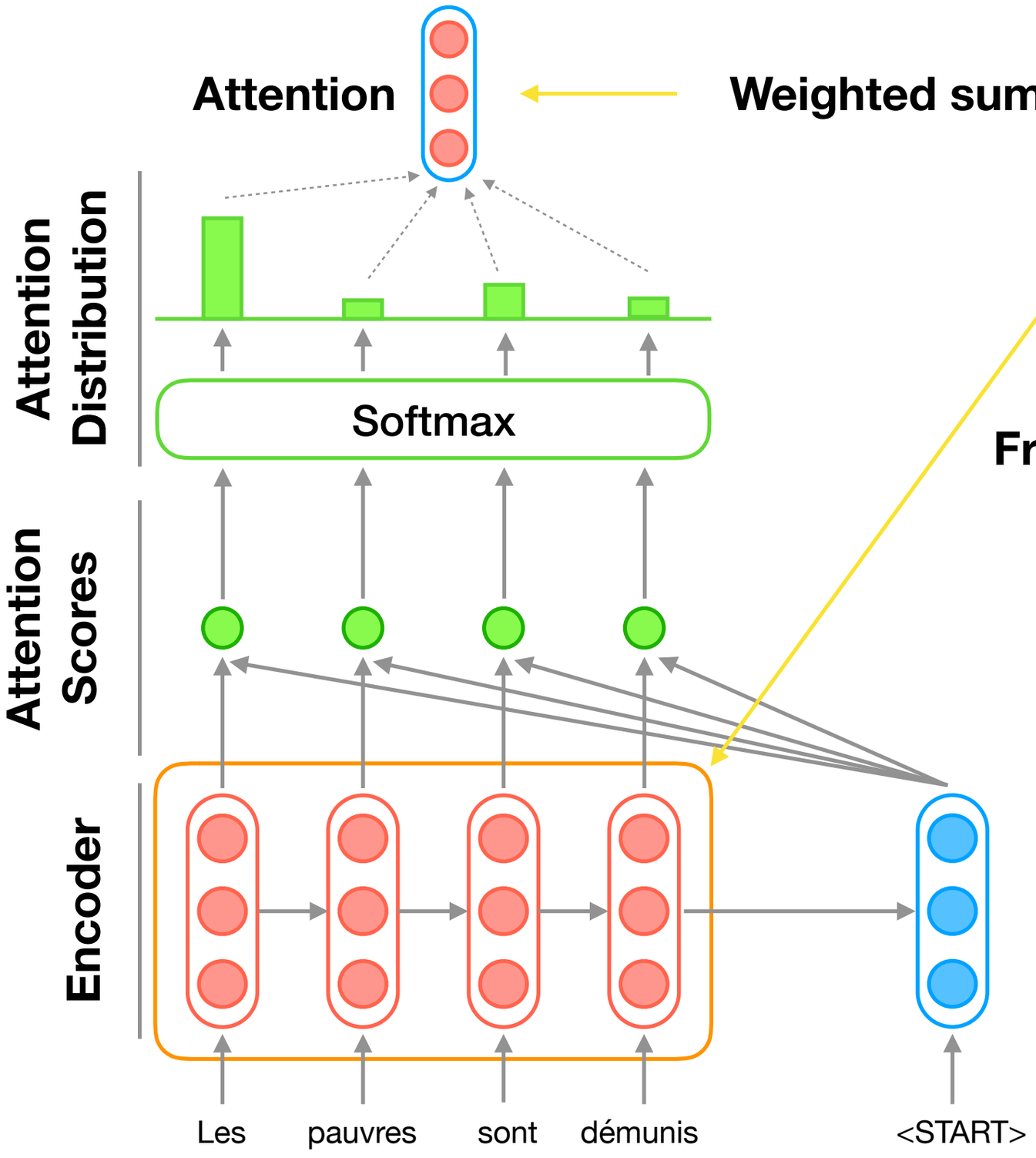
```
1 - distance.cosine(x1, x2)
```

```
0.7349837217006939
```

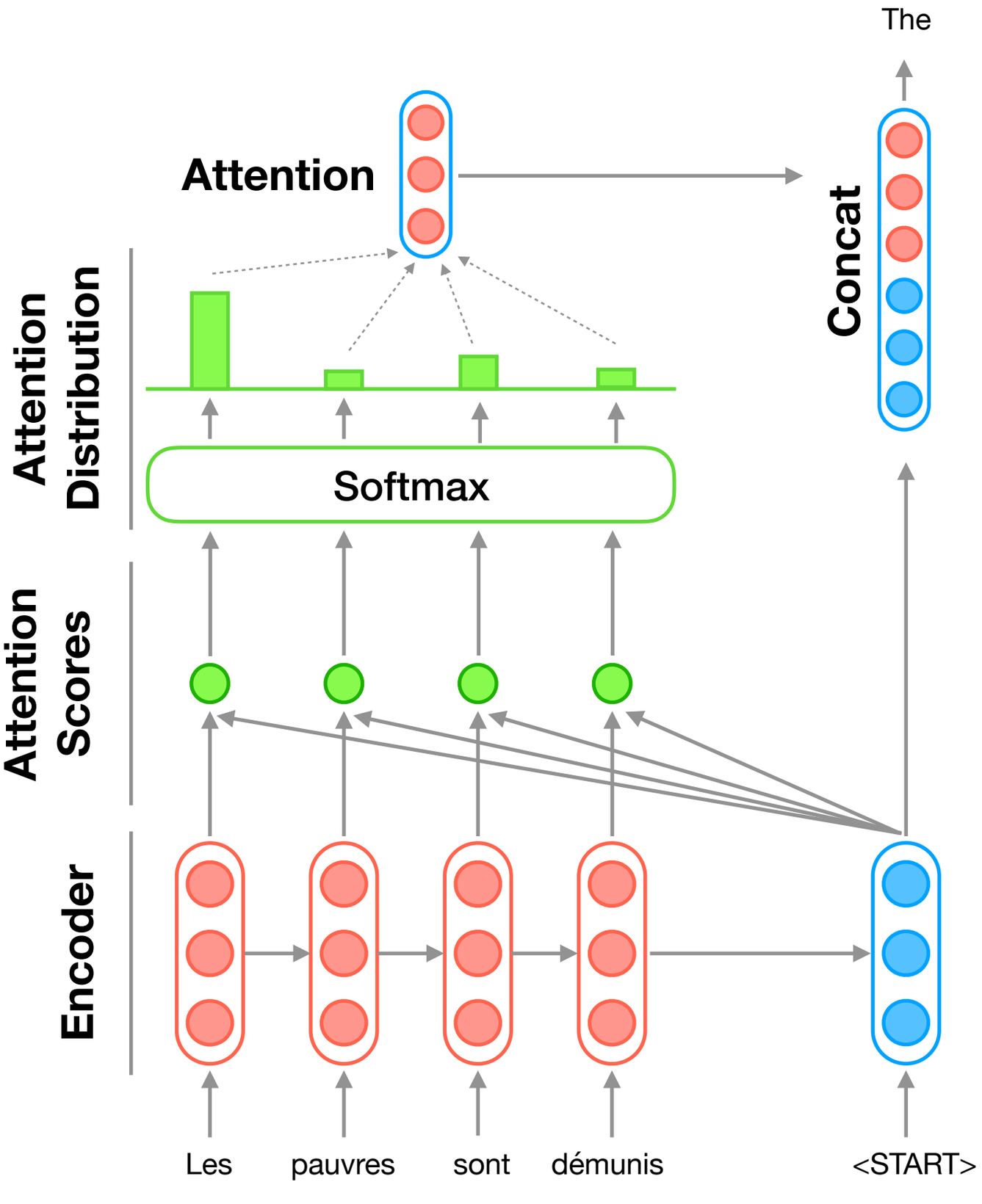
Attention


Attention
Attention Distribution
Softmax
Attention Scores
Encoder
Les
pauvres
sont
démunis
<START>

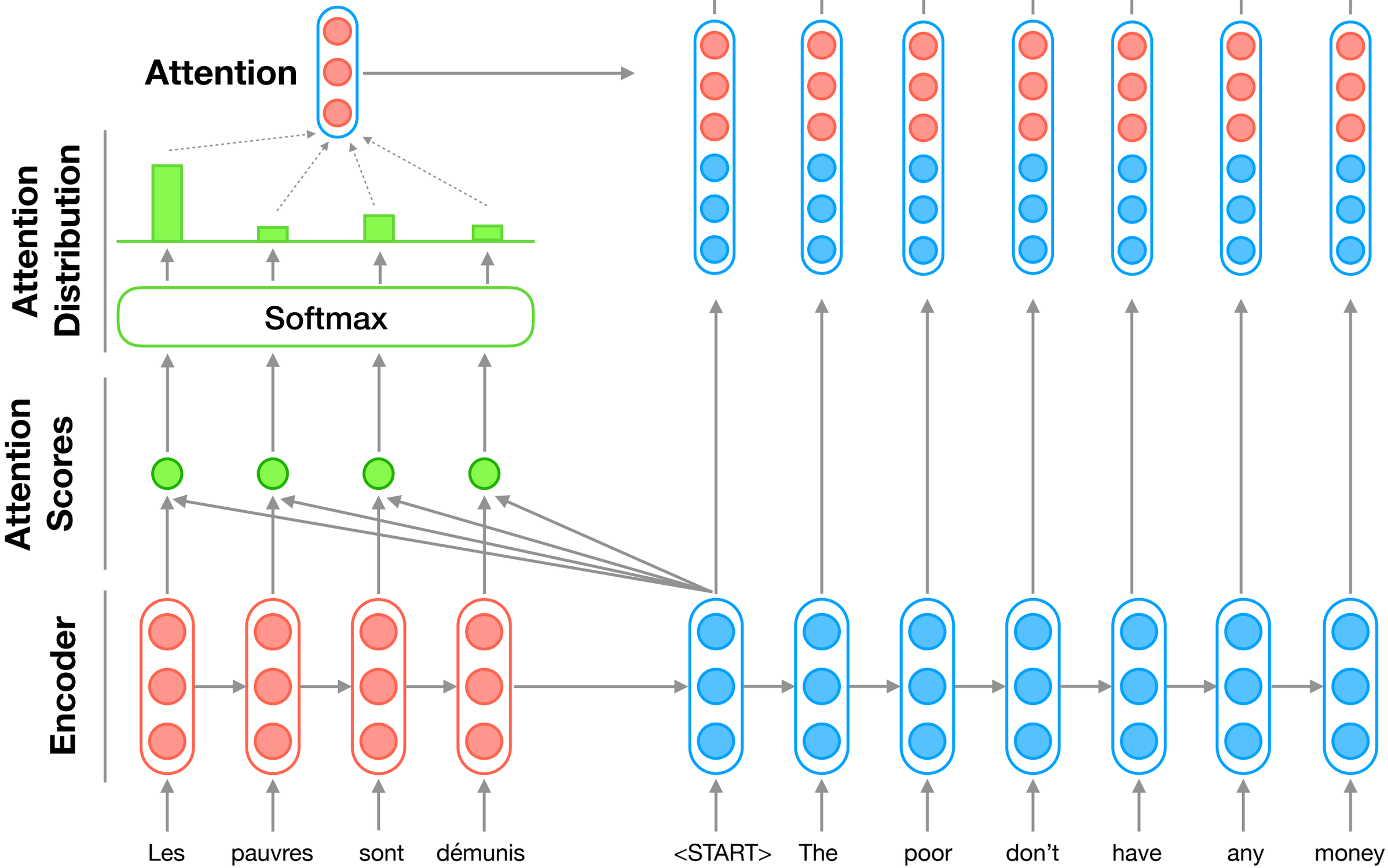
Attention


Attention
Weighted sum of encoder hidden states
Attention Distribution
Softmax
From this
Attention Scores
Encoder
Les
pauvres
sont
démunis
<START>

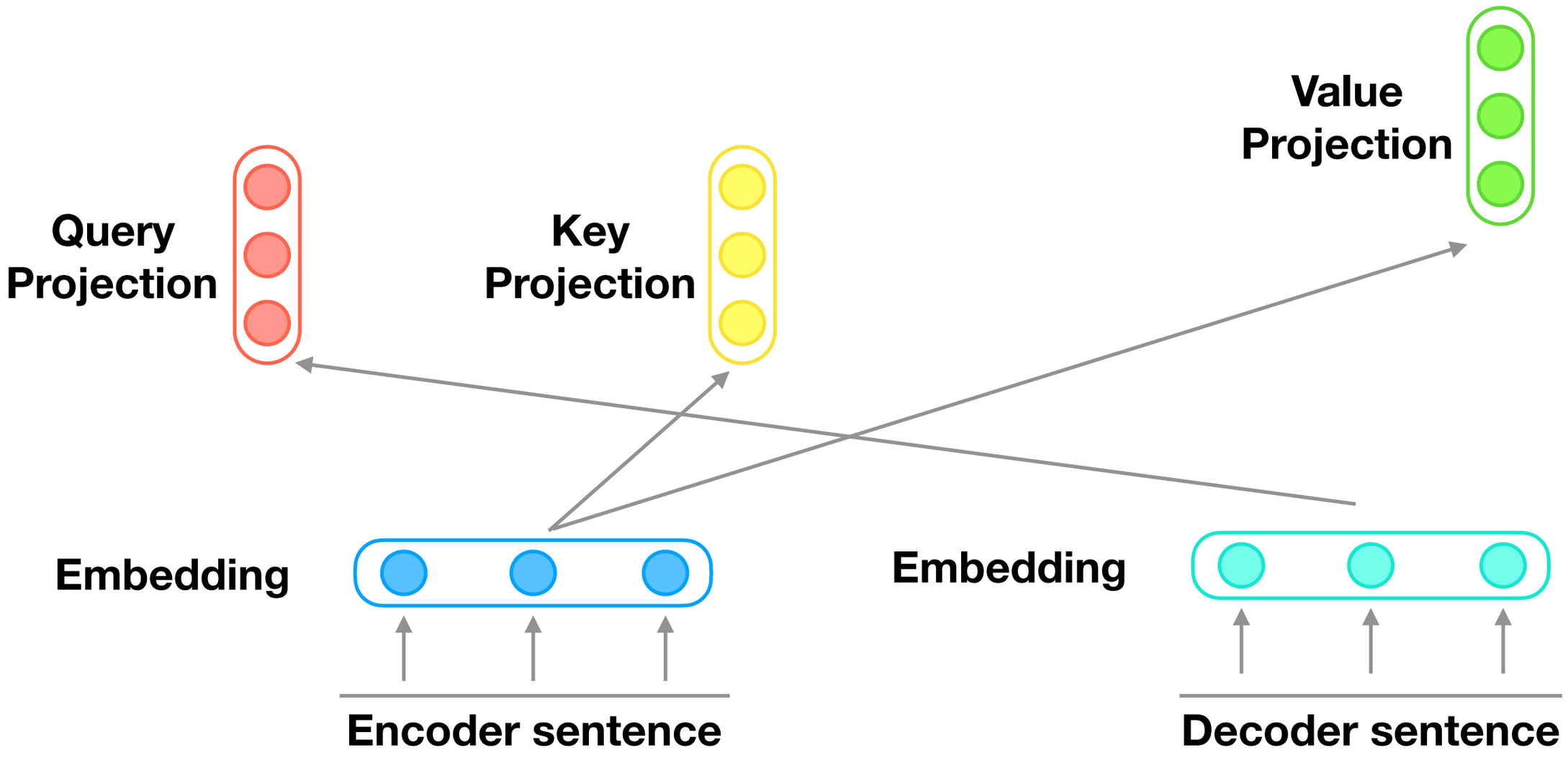
Attention


The
Attention
Concat
Attention Distribution
Softmax
Attention Scores
Encoder
Les
pauvres
sont
démunis
<START>

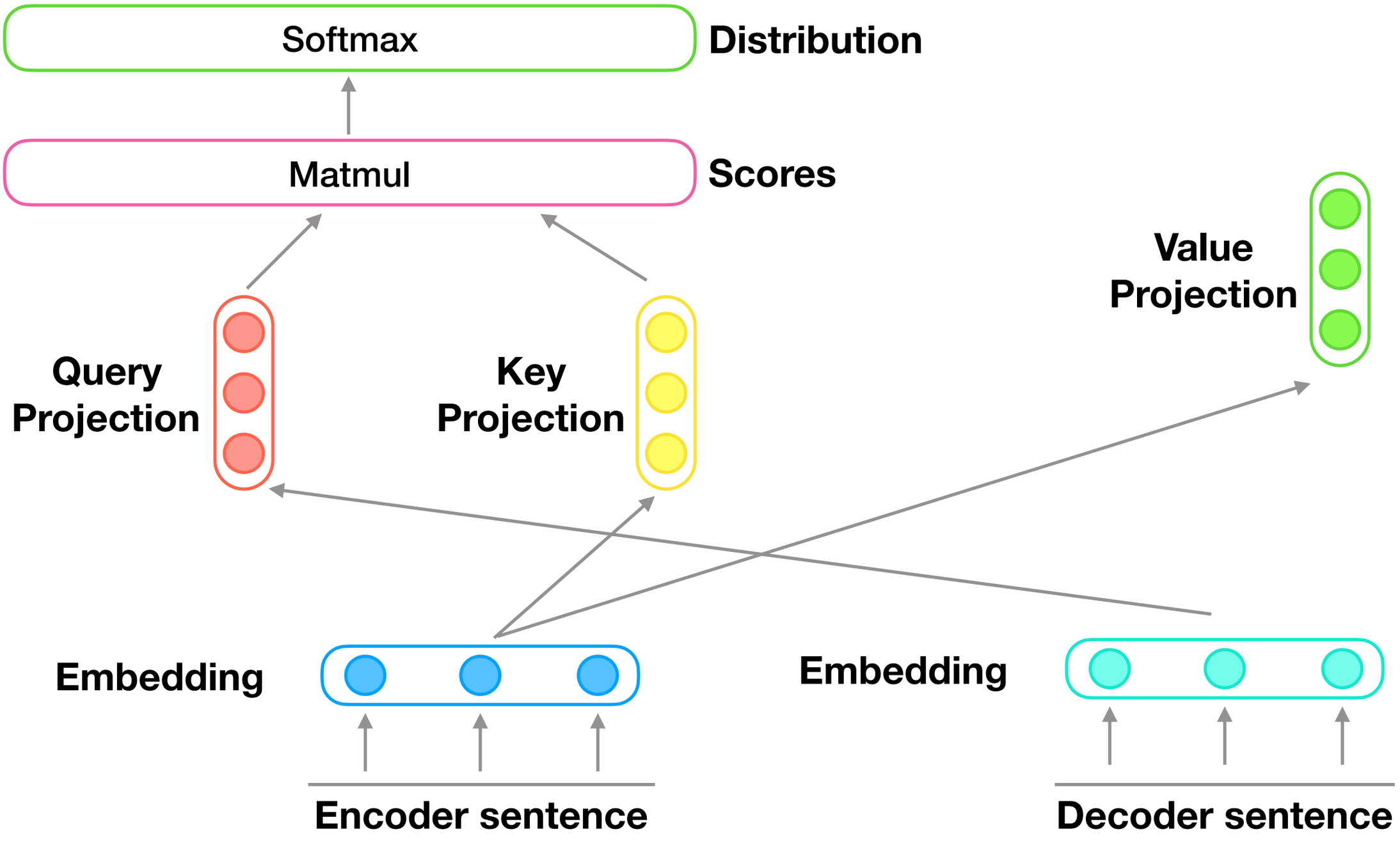
Attention


Attention
Softmax
Attention Distribution
Attention Scores
Encoder
The
poor
don’t
have
any
money
<END>
Les
pauvres
sont
démunis
<START>
The
poor
don’t
have
any
money

Attention

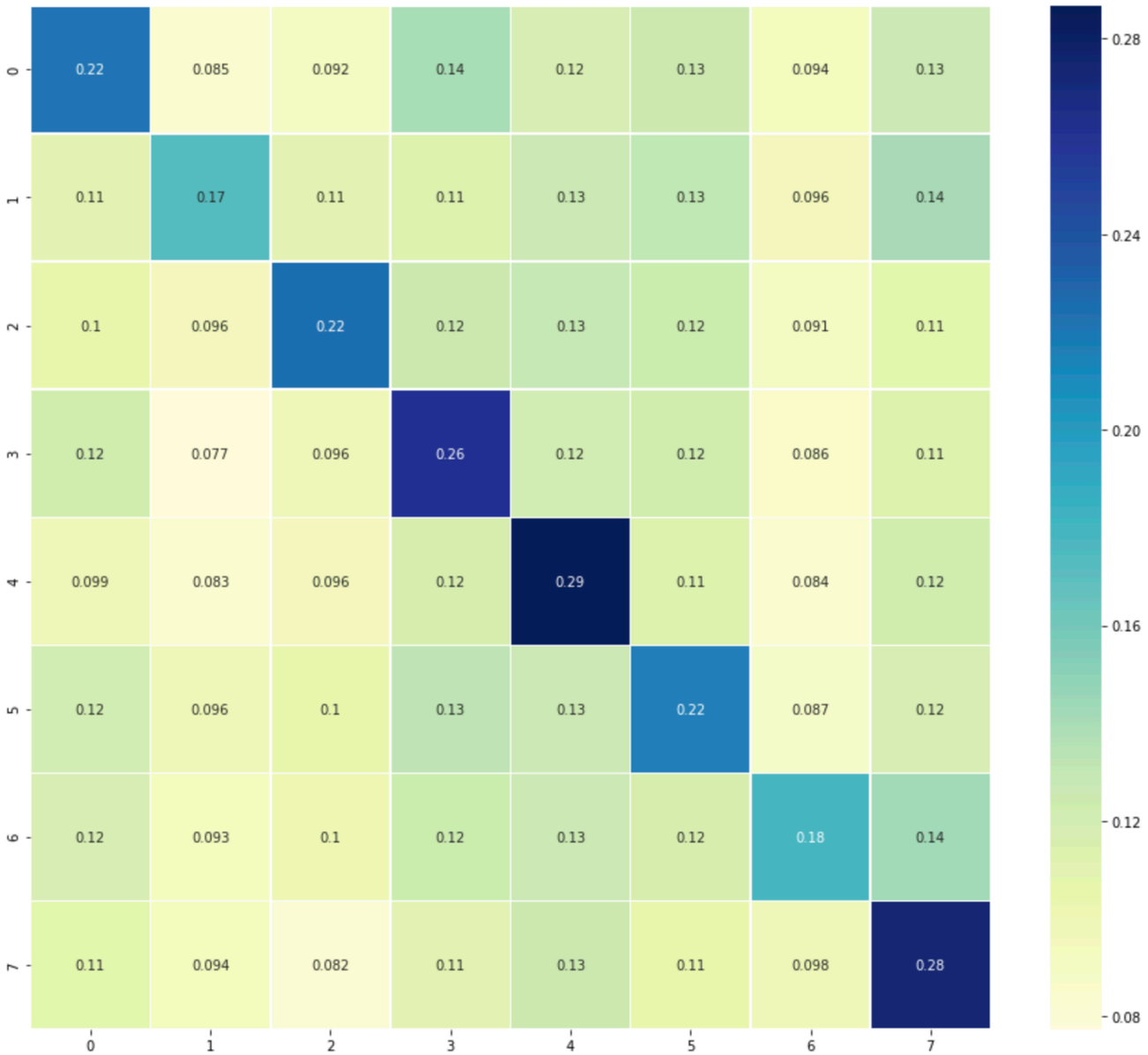

Value
Projection
Query
Projection
Key
Projection
Embedding
Embedding
Encoder sentence
Decoder sentence

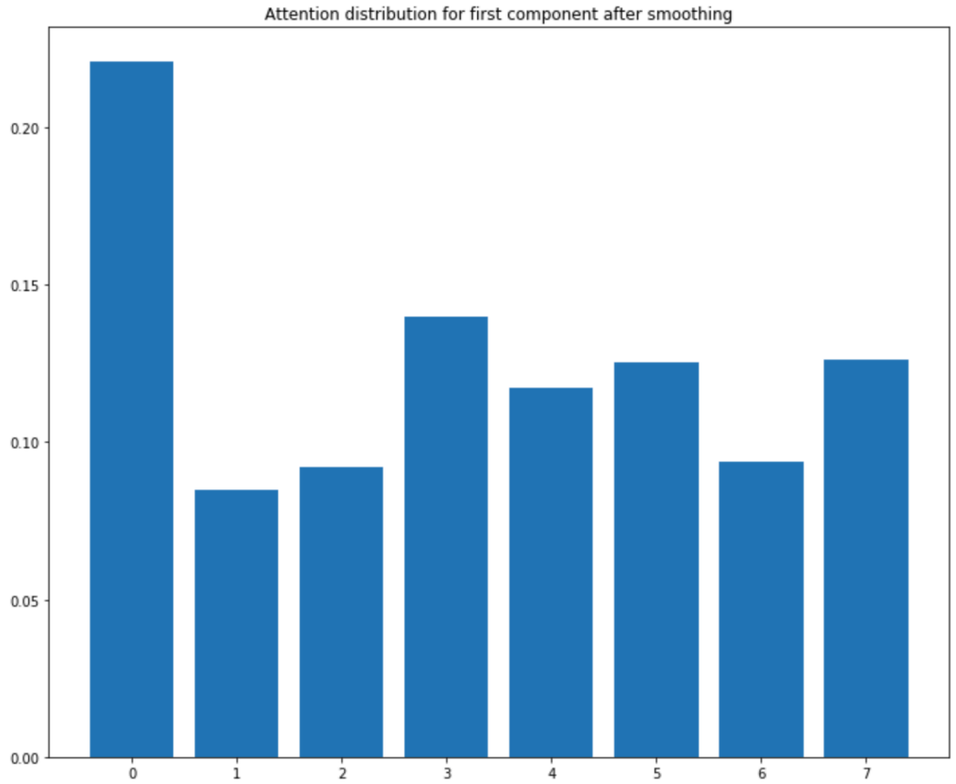
Attention


Softmax
Distribution
Matmul
Scores
Query Projection
Key Projection
Value Projection
Embedding
Embedding
Encoder sentence
Decoder sentence

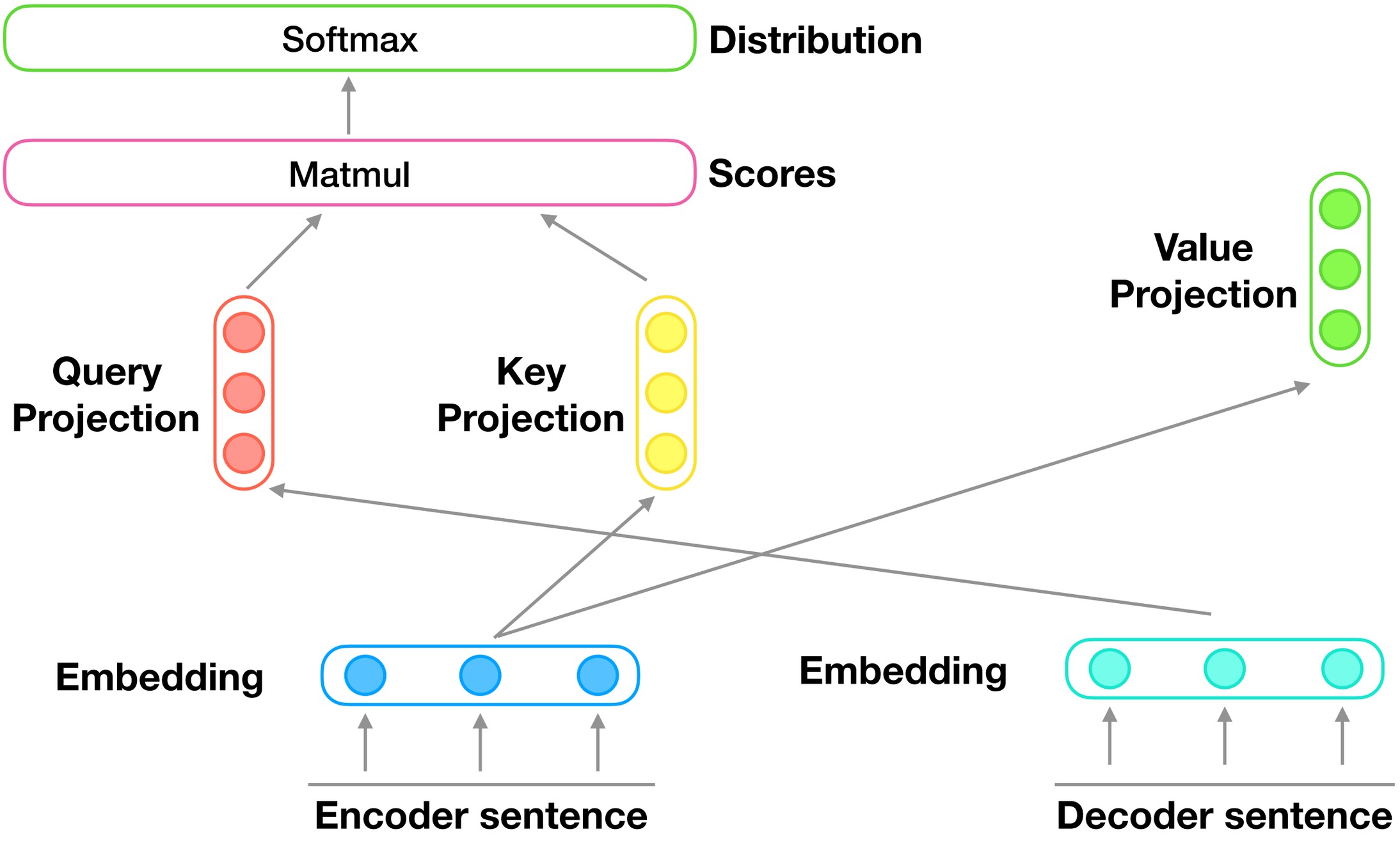
Attention

Attention Distribution
Sequence length = 8

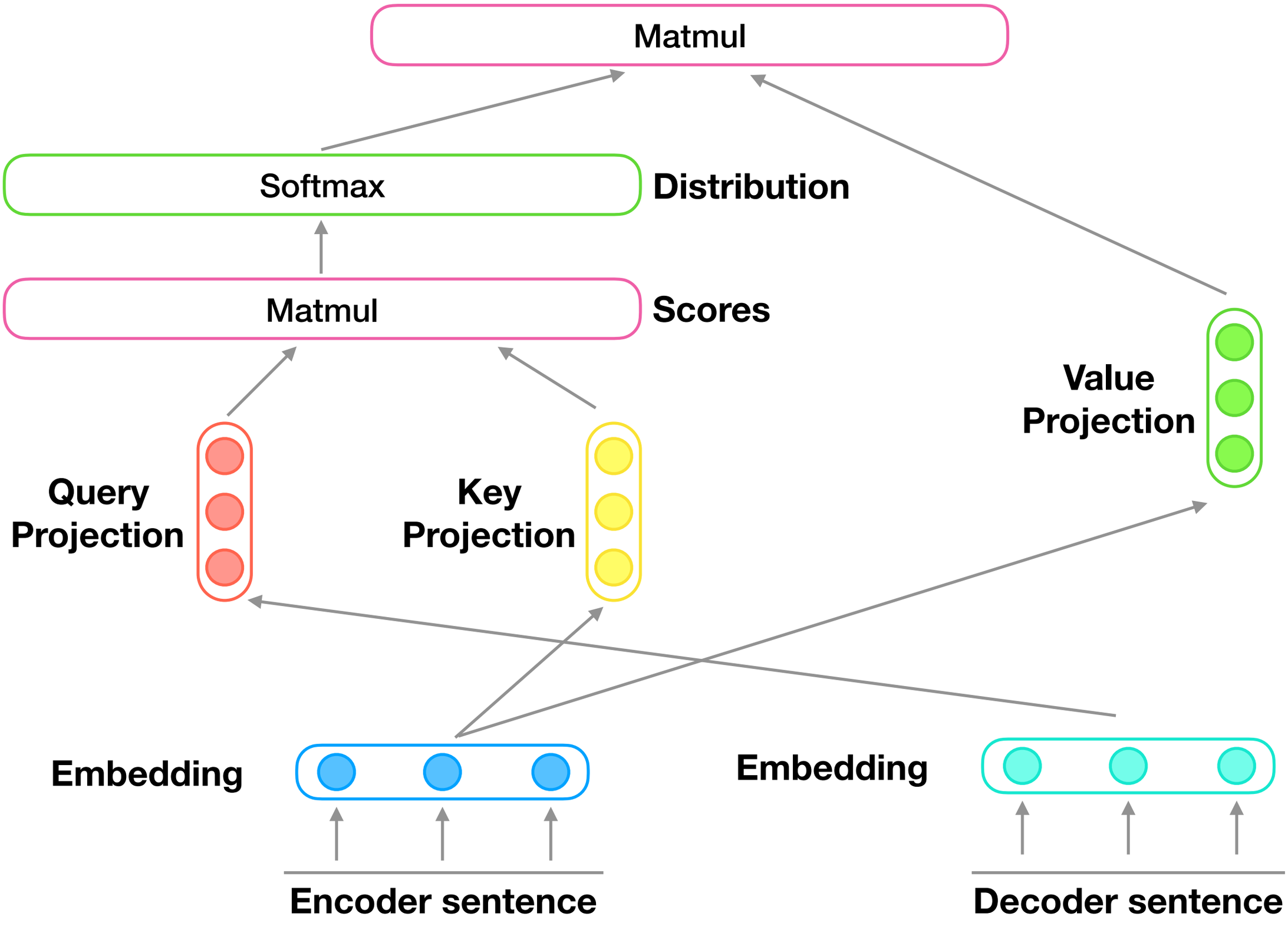




Attention

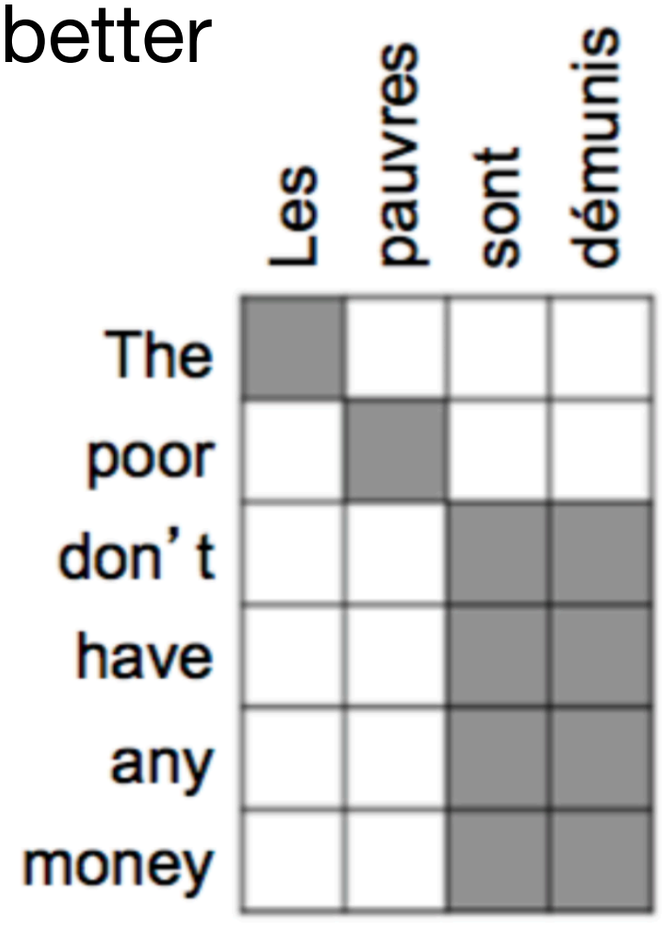

Softmax
Distribution
Matmul
Scores
Query Projection
Key Projection
Value Projection
Embedding
Embedding
Encoder sentence
Decoder sentence

Attention


Matmul
Softmax
Distribution
Matmul
Scores
Value
Projection
Query
Projection
Key
Projection
Embedding
Embedding
Encoder sentence
Decoder sentence

Attention

- Work with long dependencies
- Significantly improves NMT performance
- Solves bottleneck problem
- Attention helps to do your language task better



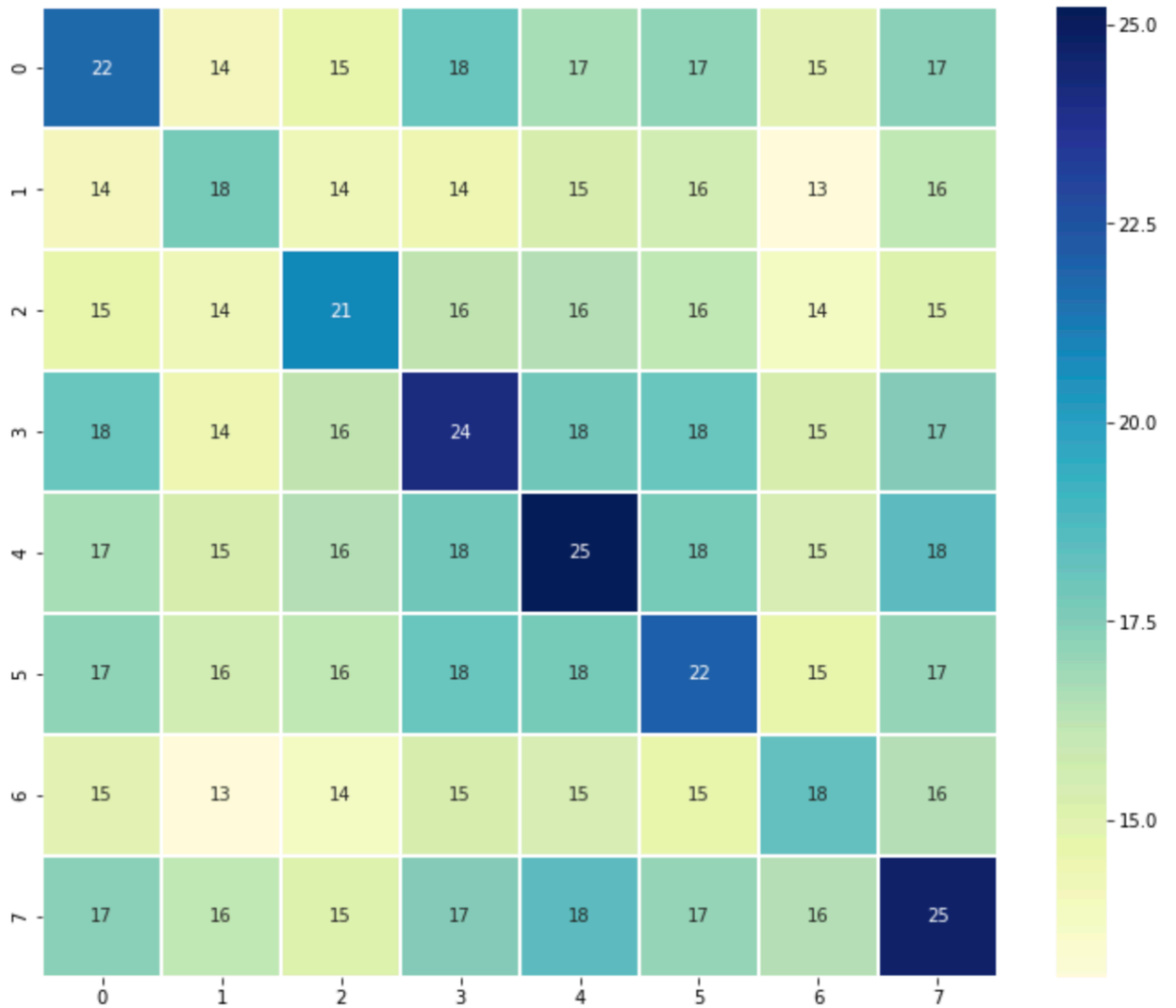
Self-Attention

What if we apply attention to yourself?

```
attention_scores = torch.matmul(x, x.t())
```

x.shape == (sequence_length, embedding_dim)

softmax(



)

Self-Attention

What if we apply attention to yourself?

```
attention_distribution = torch.nn.functional.softmax(attention_scores, dim=1)
```

x.shape == (sequence_length, embedding_dim)

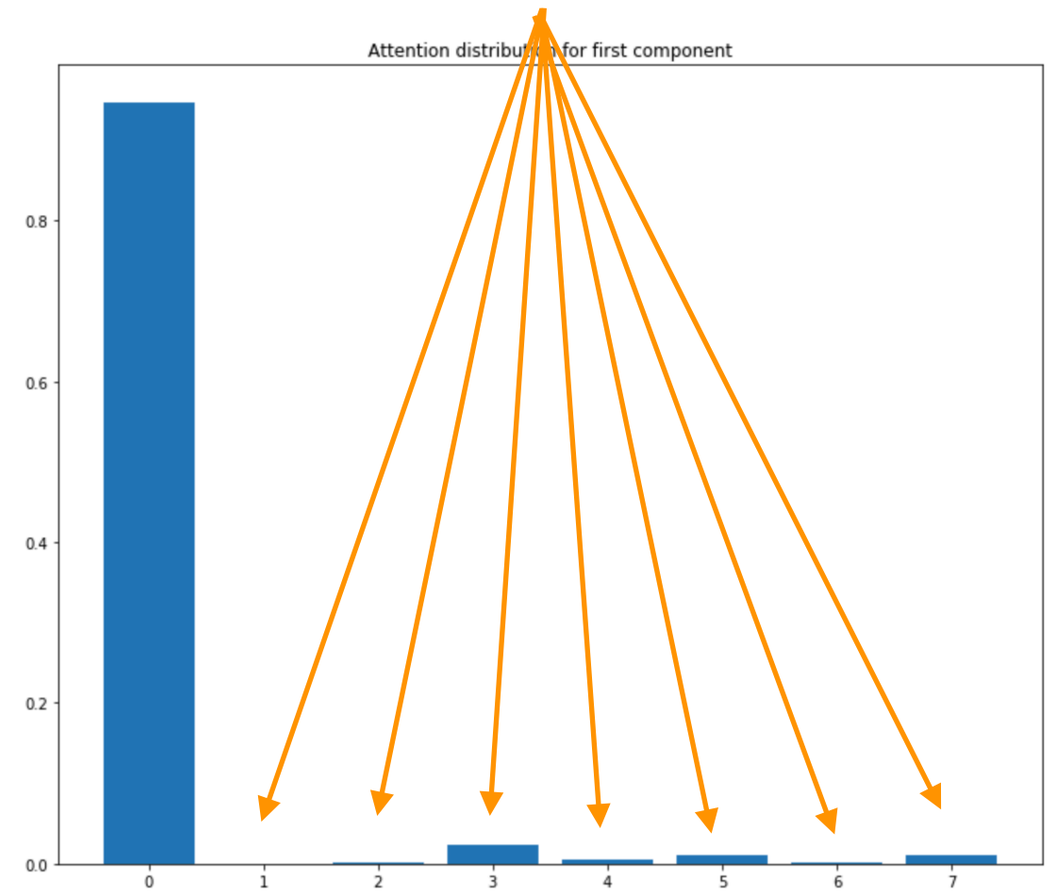
Scaled Dot-Product Attention

Scaling

Smoothing for faster convergence

Attention distribution

Extremely small gradients

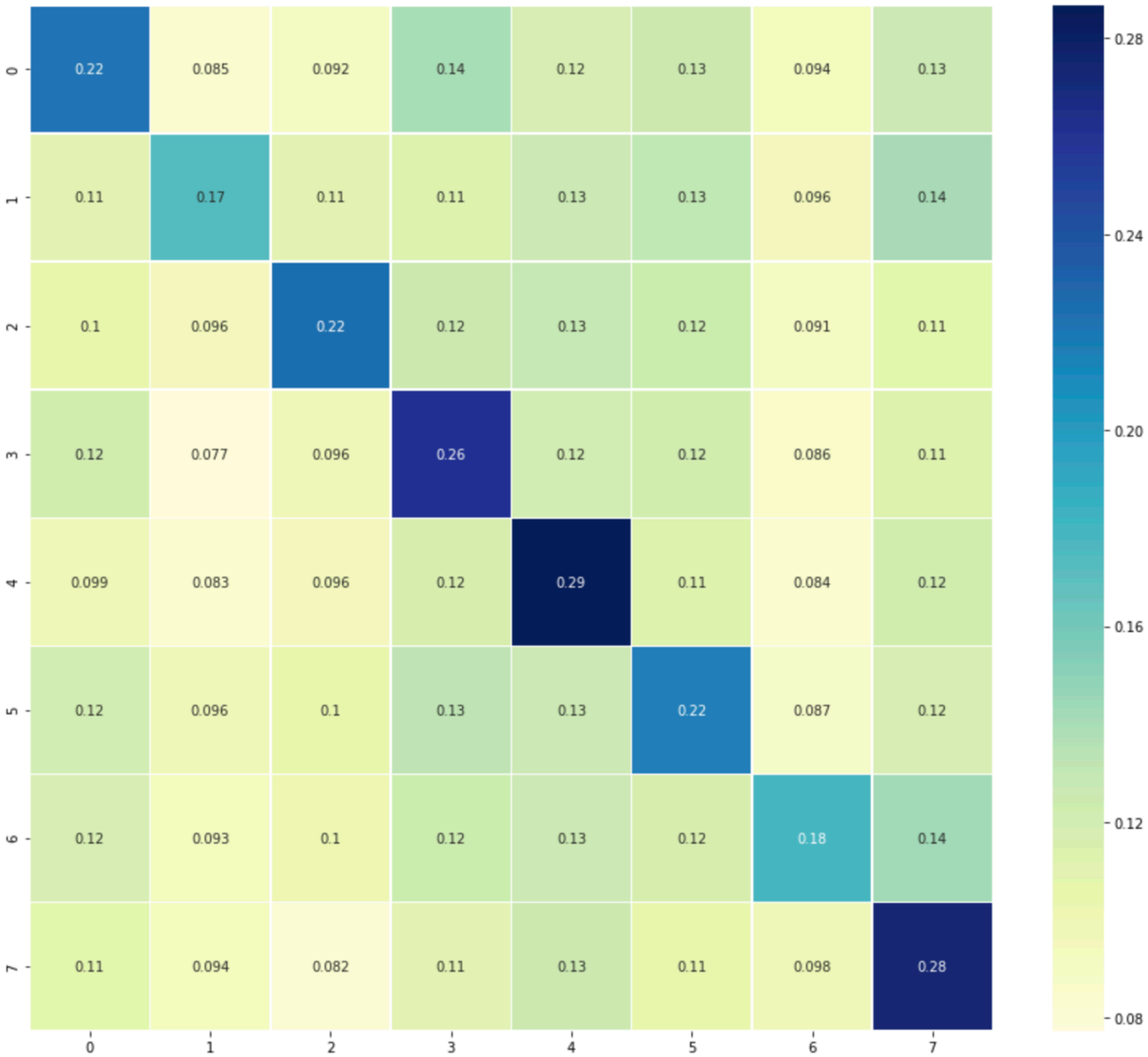


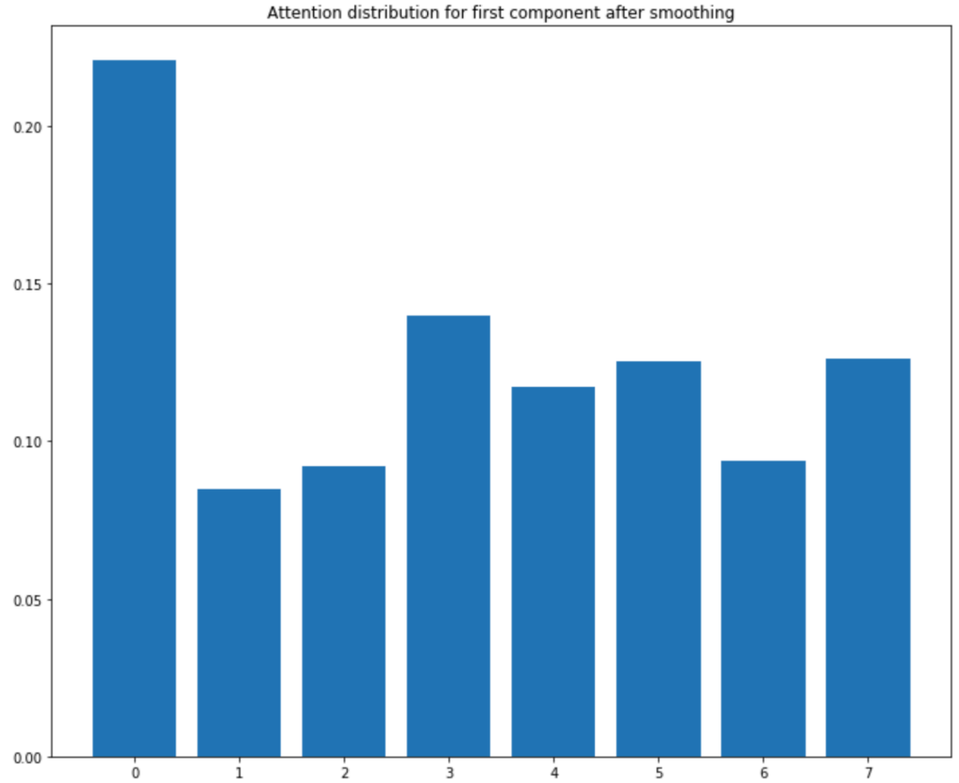
Scaled Dot-Product Attention

Scaling

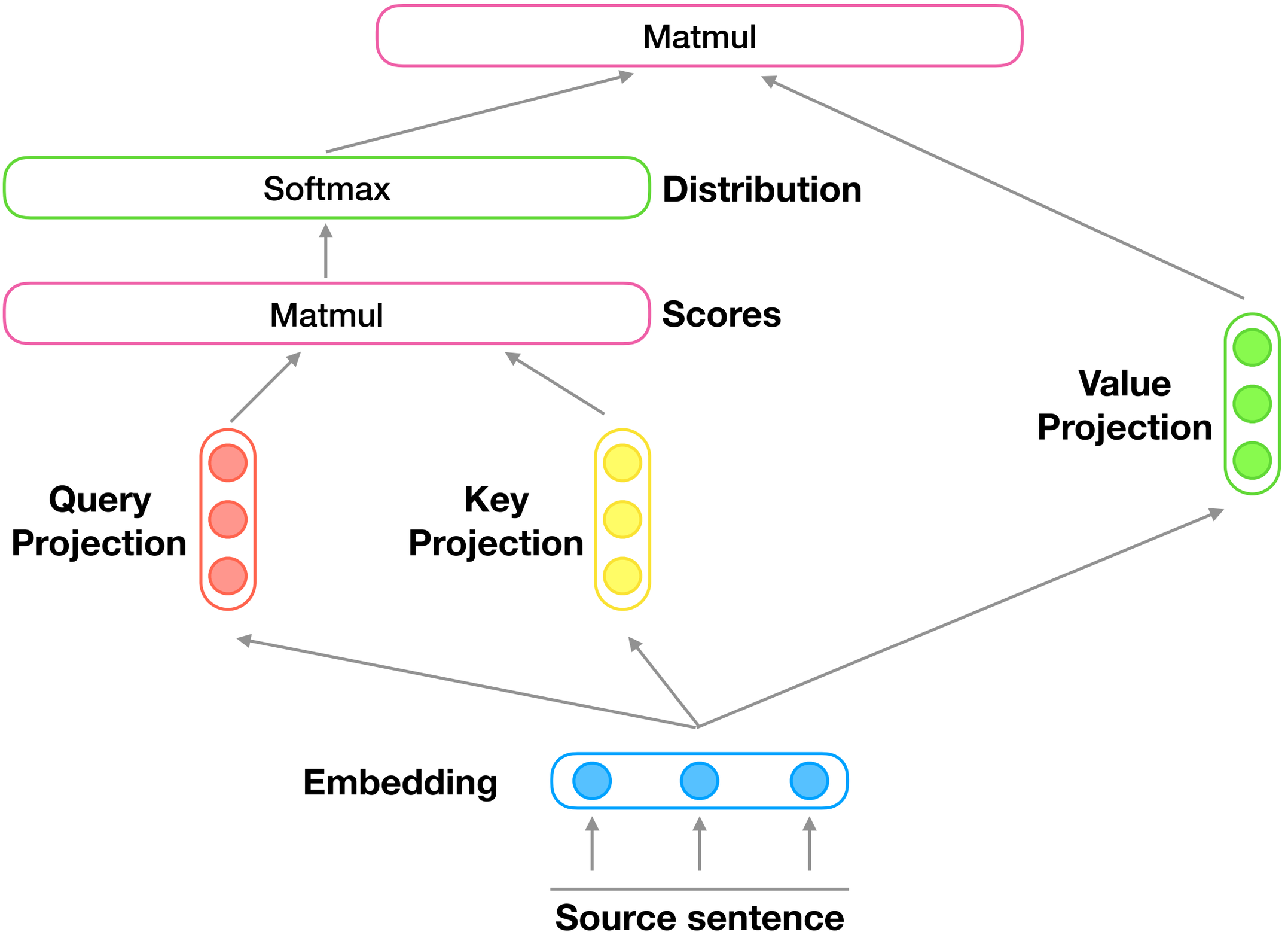
Smoothing for faster convergence

Attention distribution after scaling

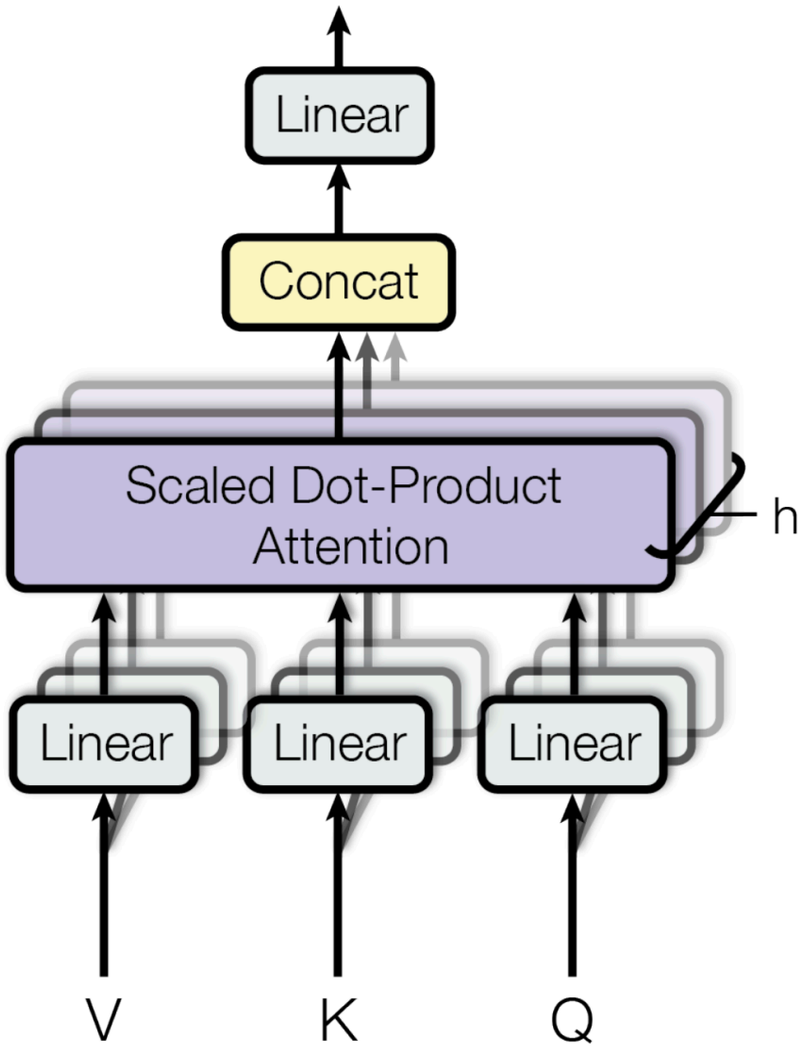




Scaled Dot-Product Attention

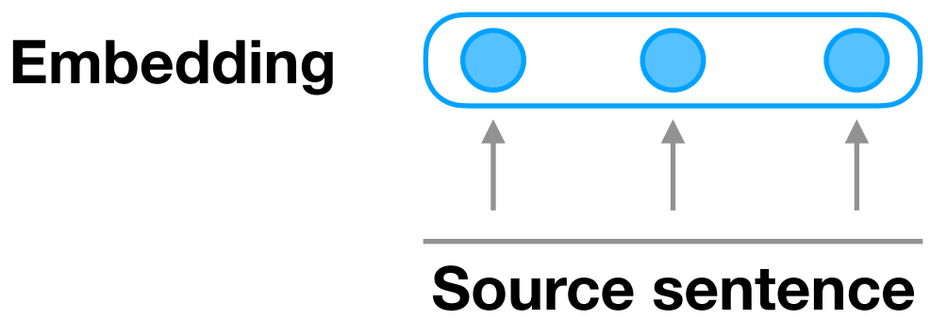


Multi-Head Attention

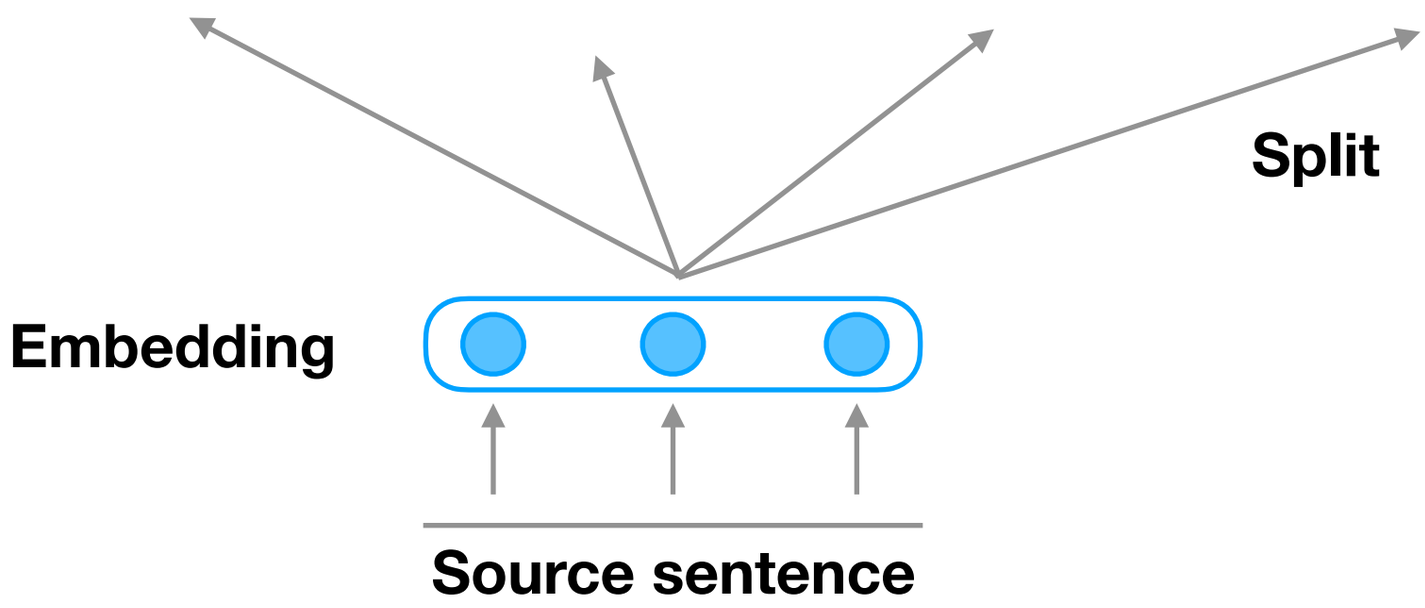


- Same things look different and we learning to look differently
- It’s like convolution filters
- After split to different heads and learning something we need concatenate it
- And mixed up heads by another linear projection

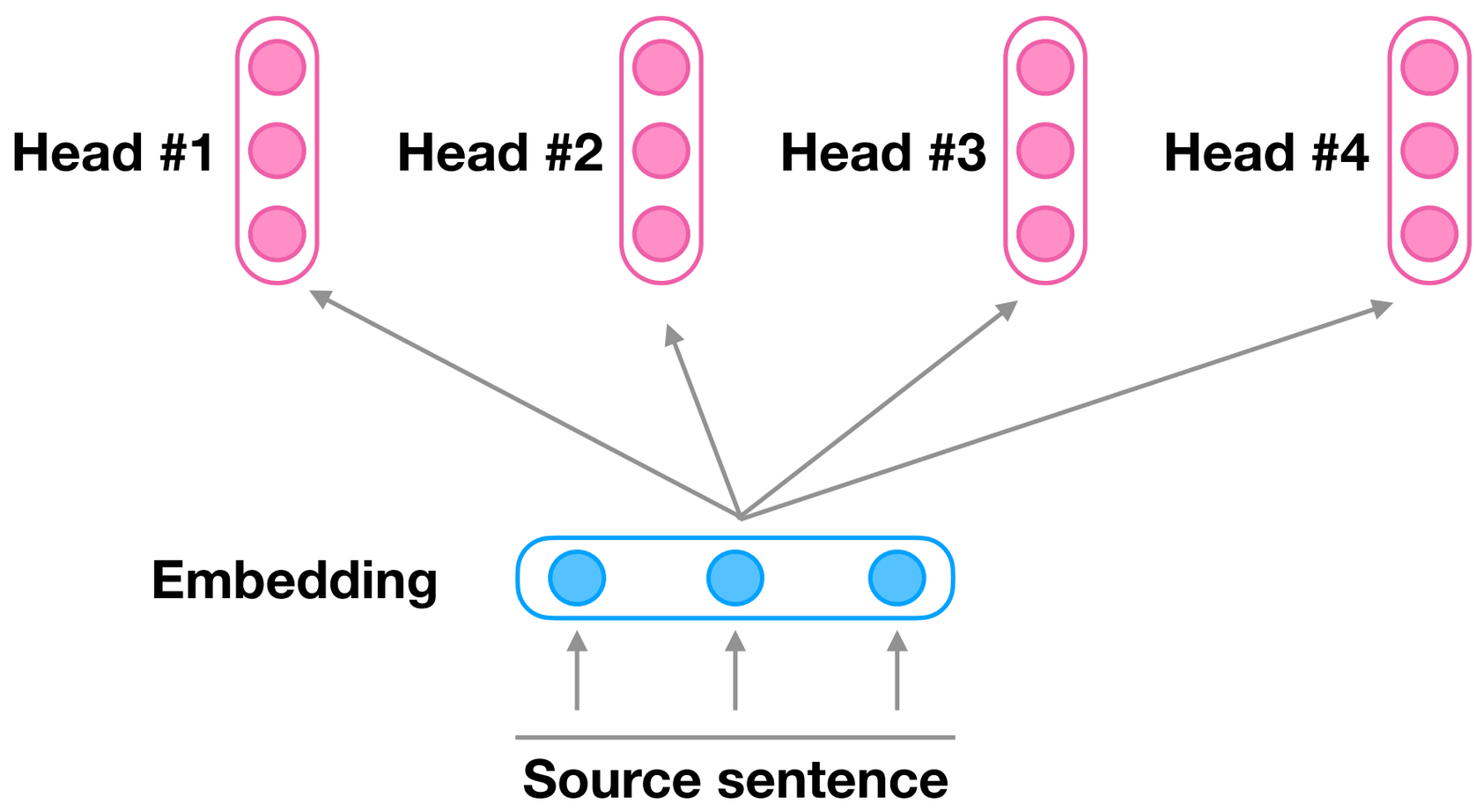
Multi-Head Attention



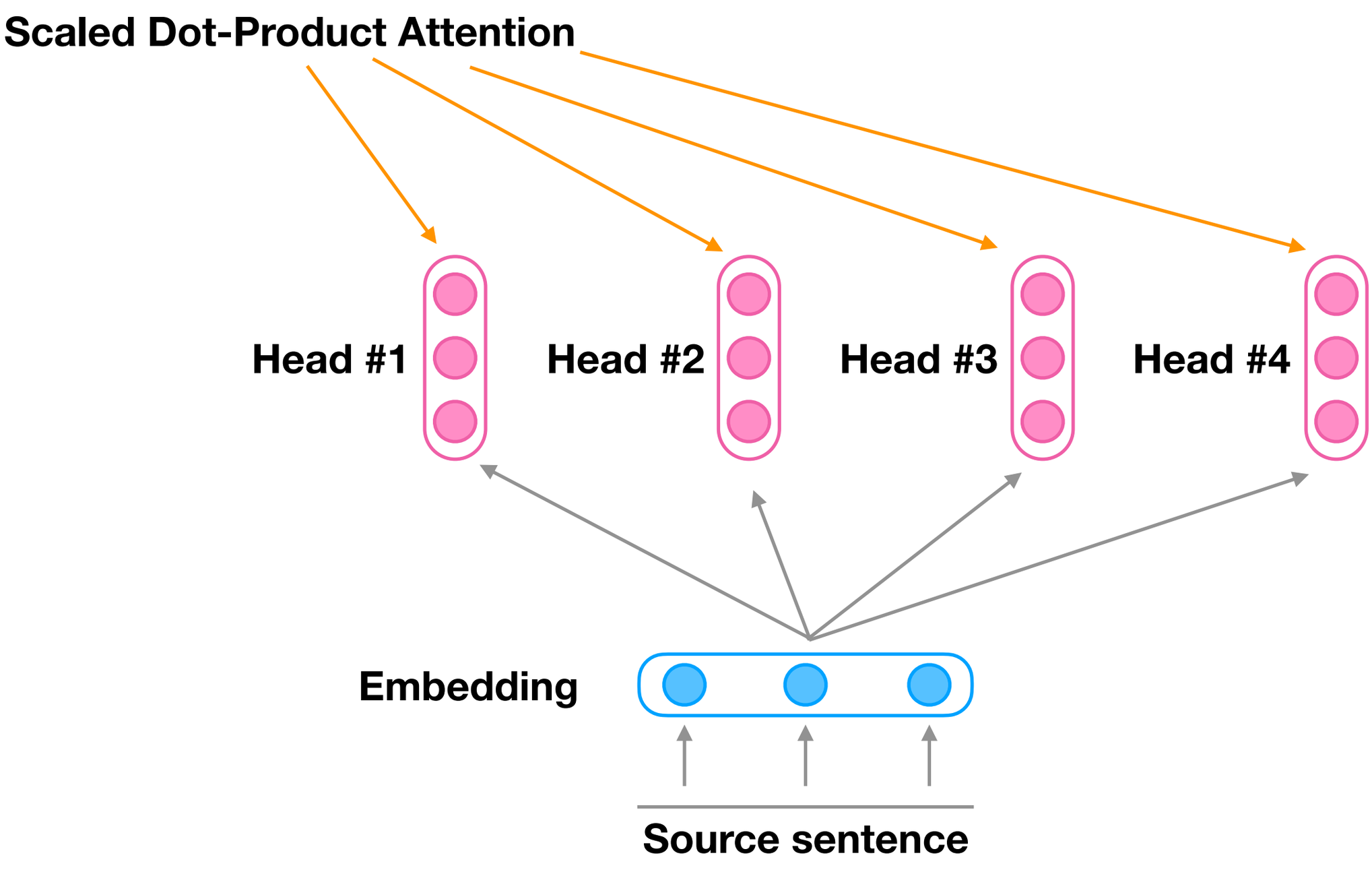
Multi-Head Attention



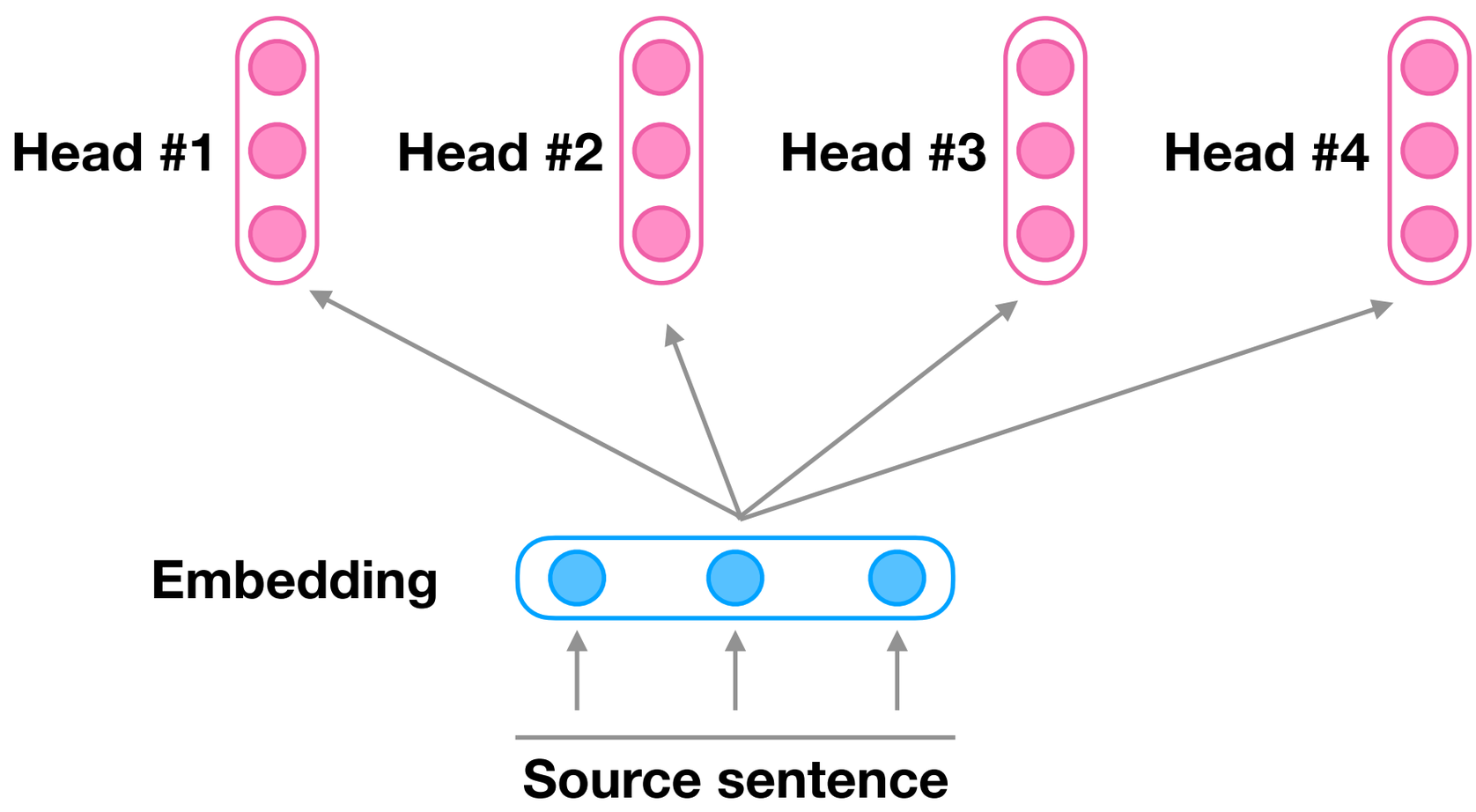
Multi-Head Attention



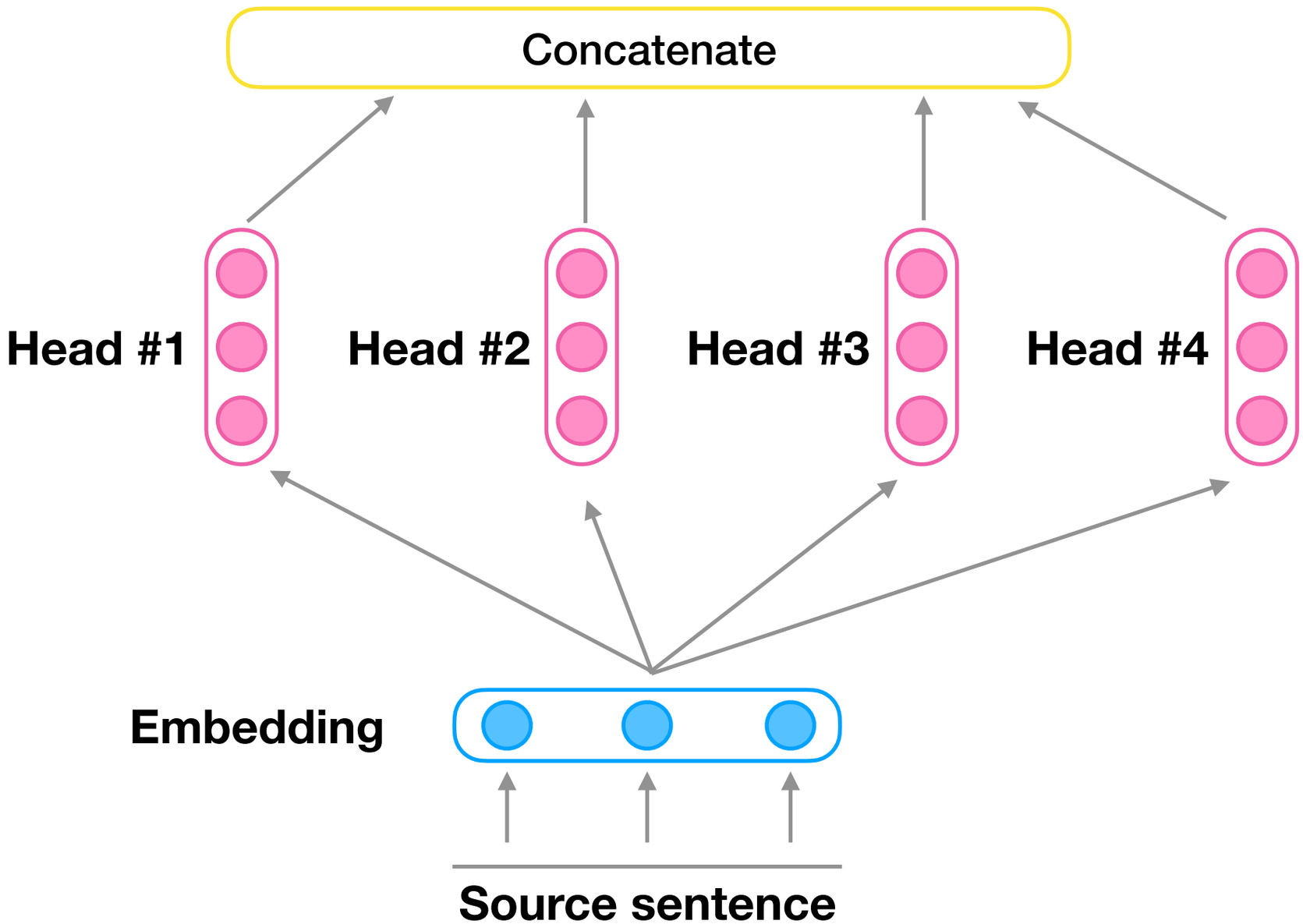
Multi-Head Attention



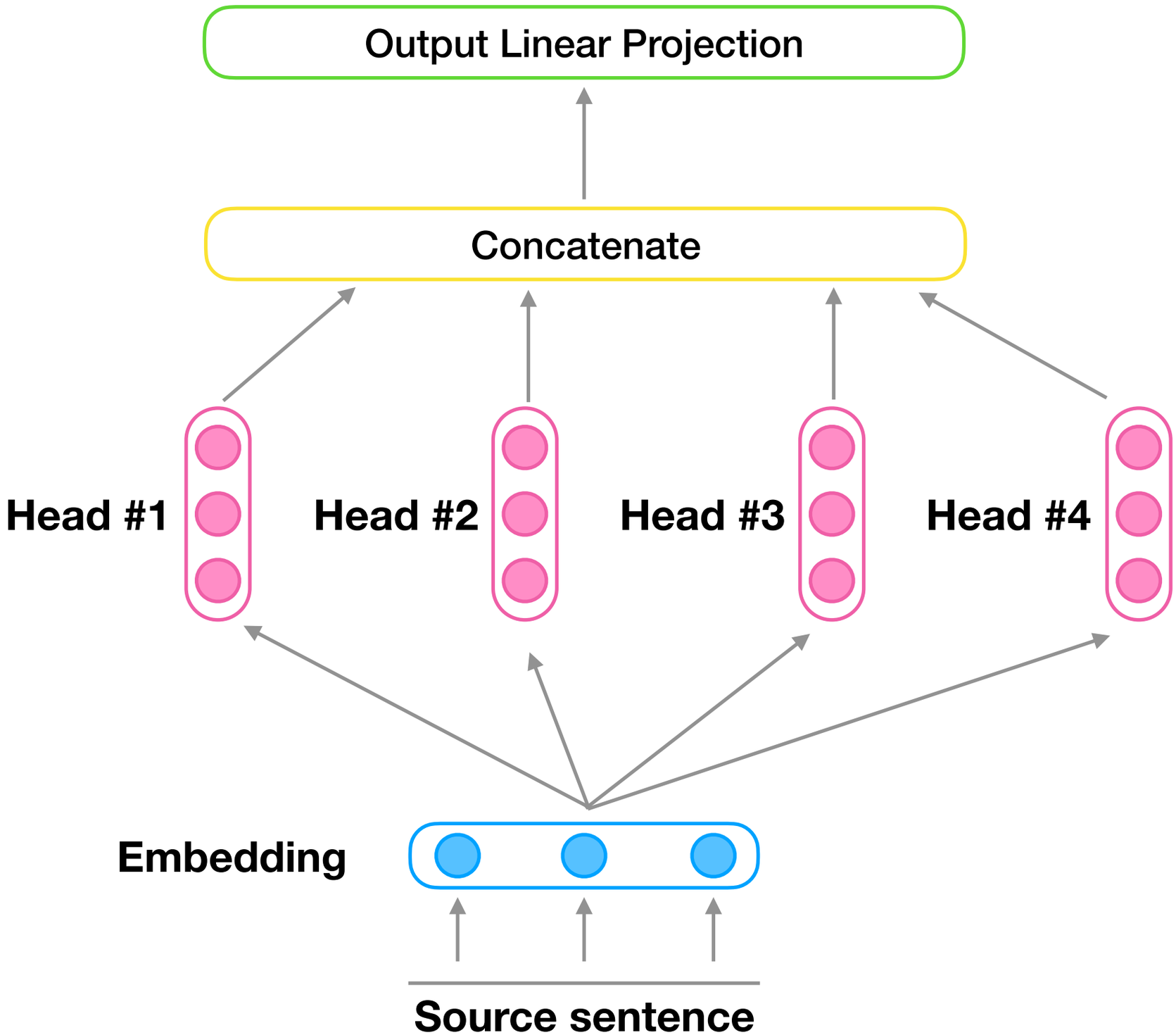
Multi-Head Attention



Multi-Head Attention

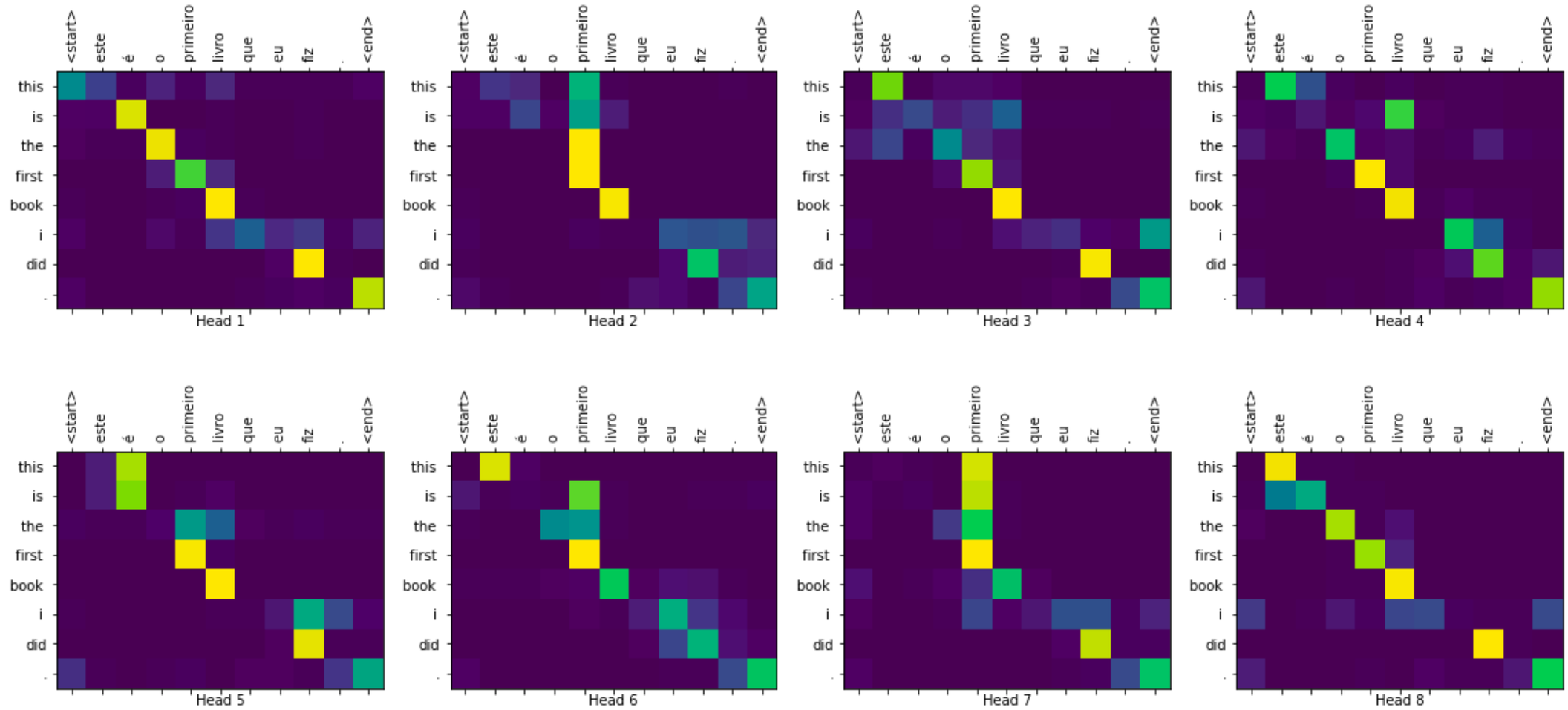


Multi-Head Attention



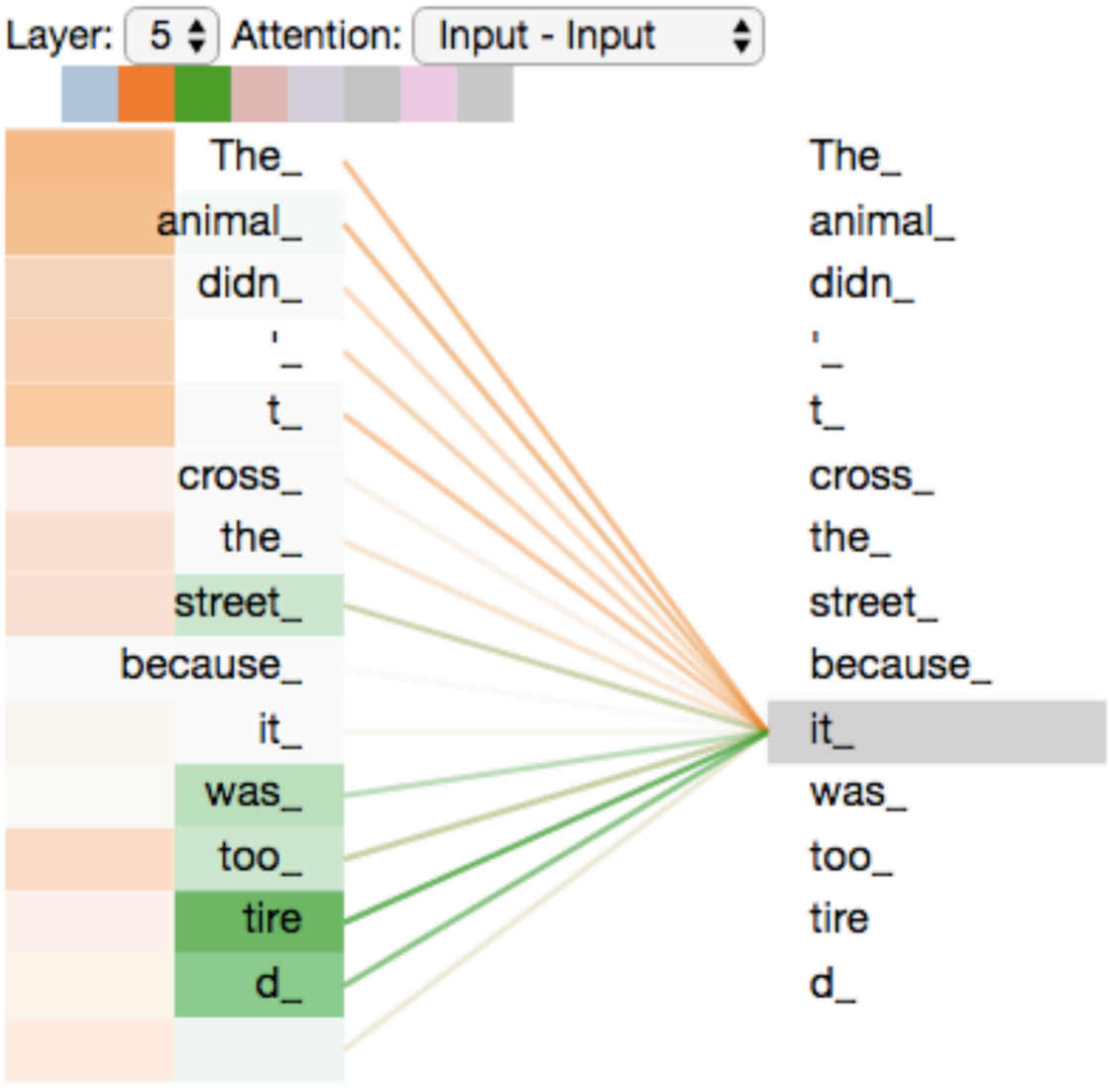
Multi-Head Attention

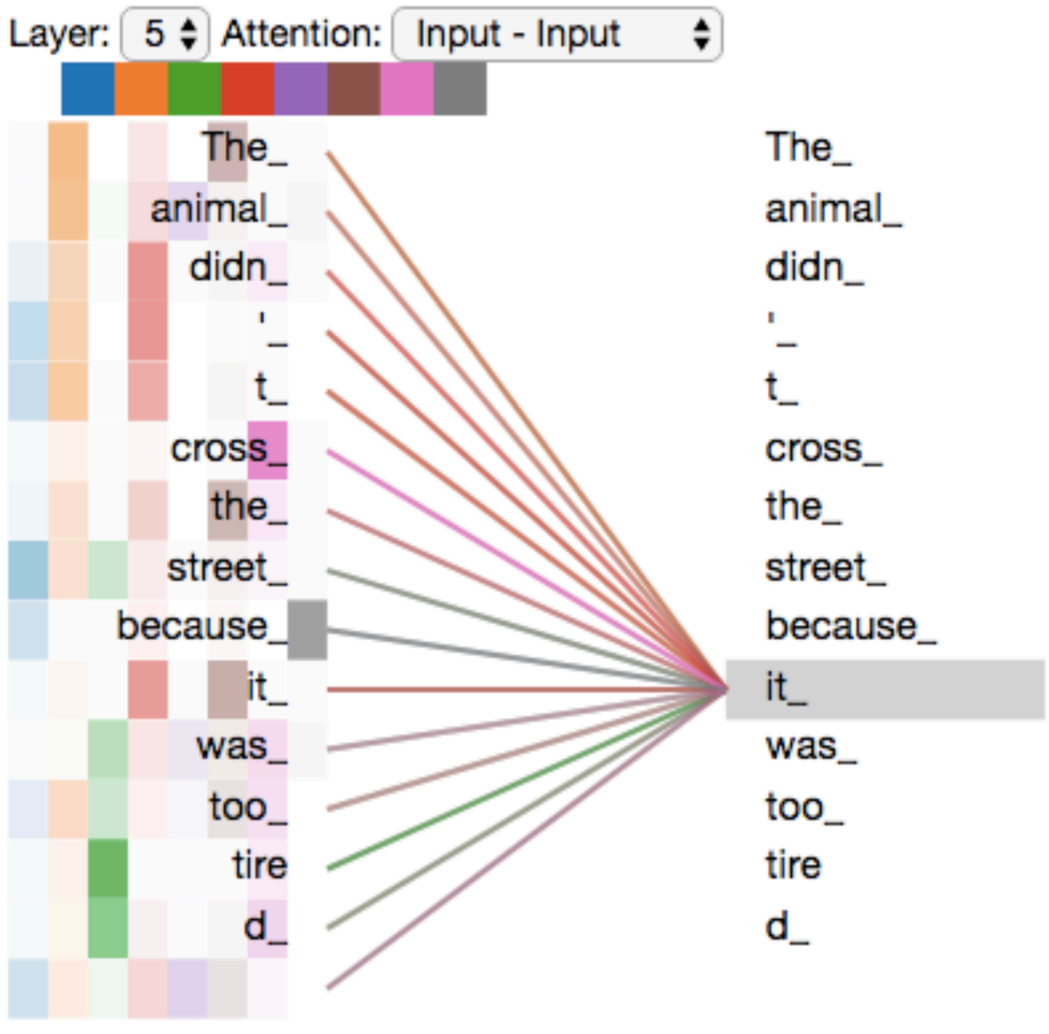
Attention Heads Visualization



Multi-Head Attention

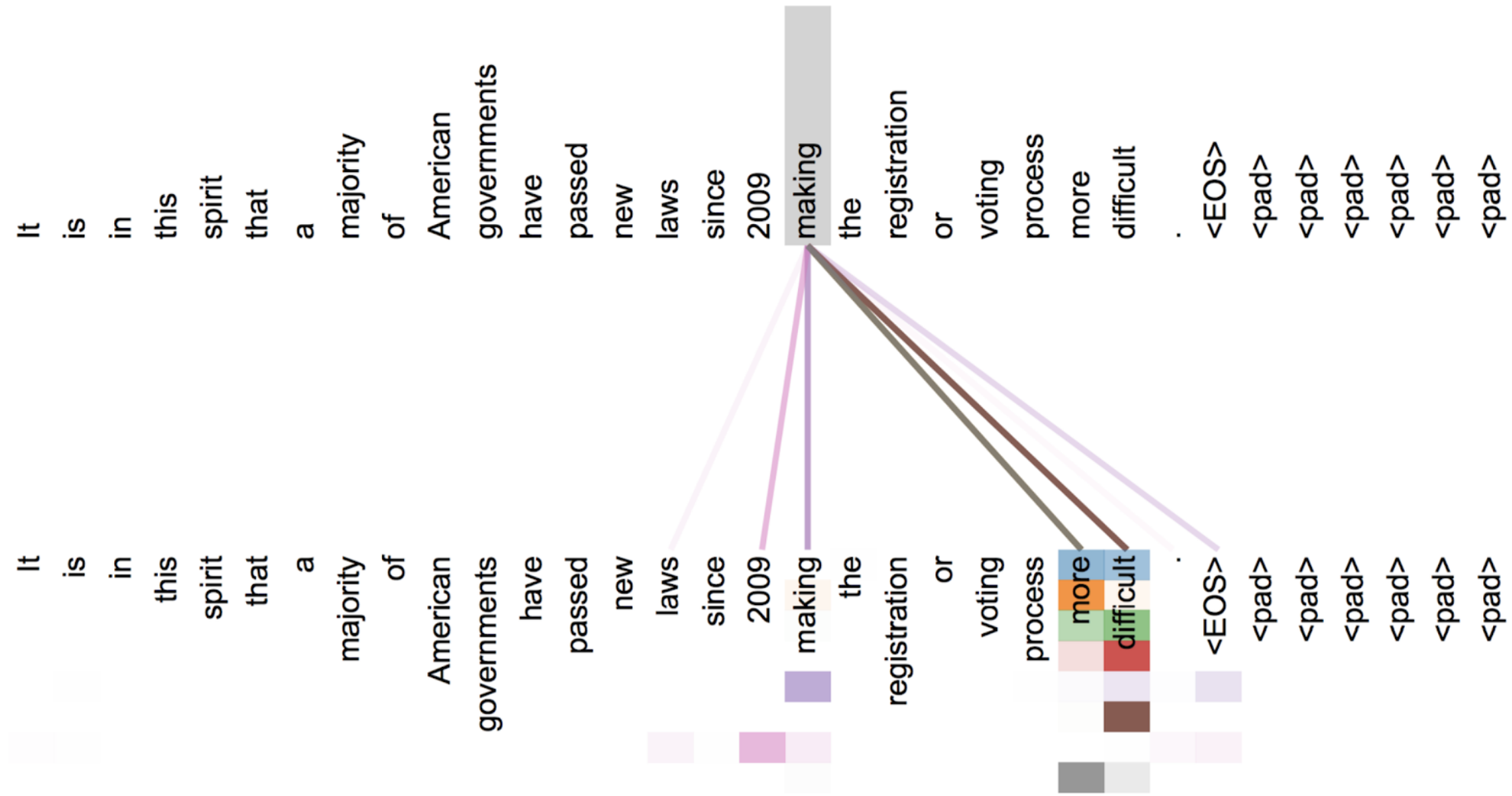
Attention Heads Visualization





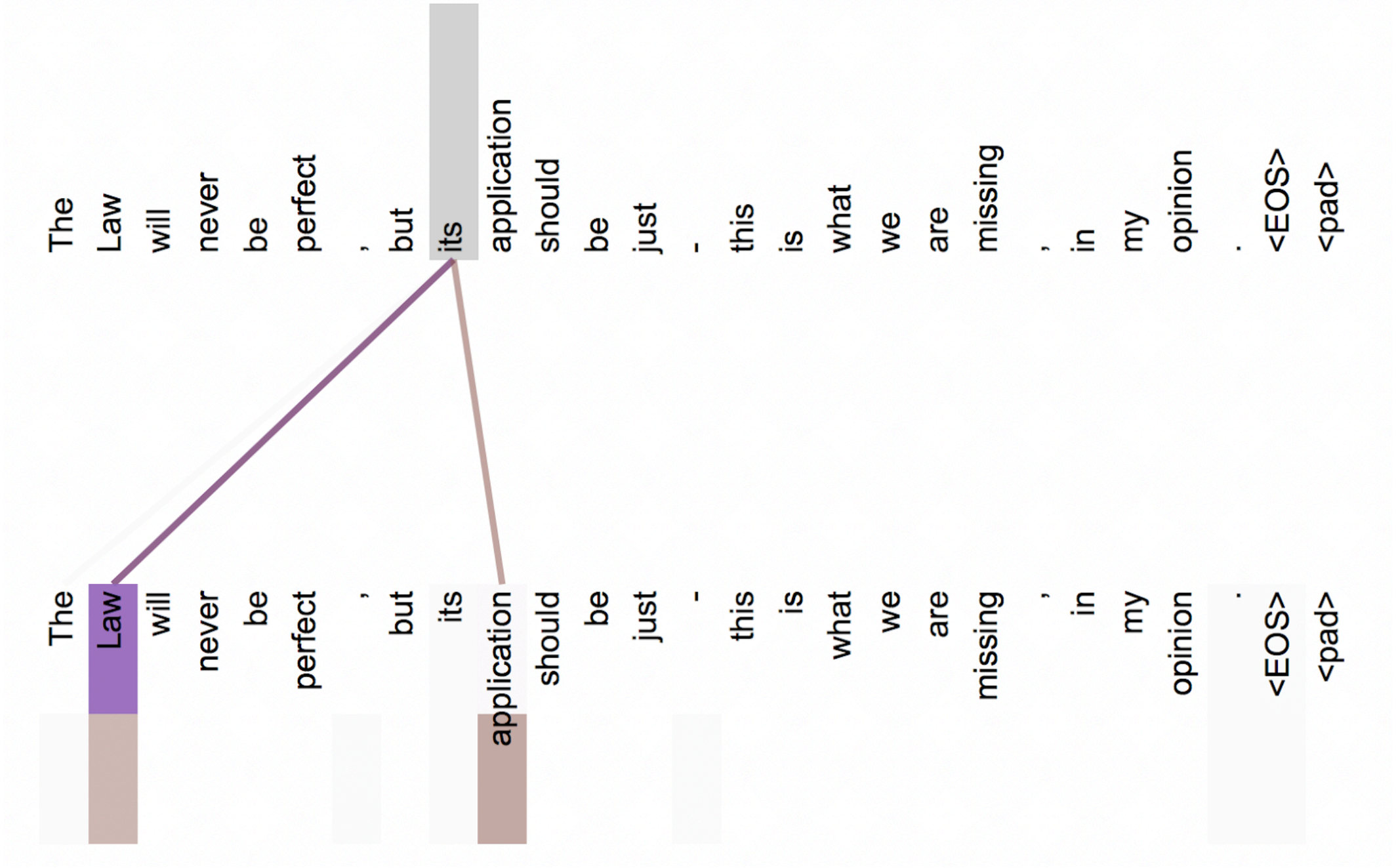
Multi-Head Attention

Attention Heads Visualization

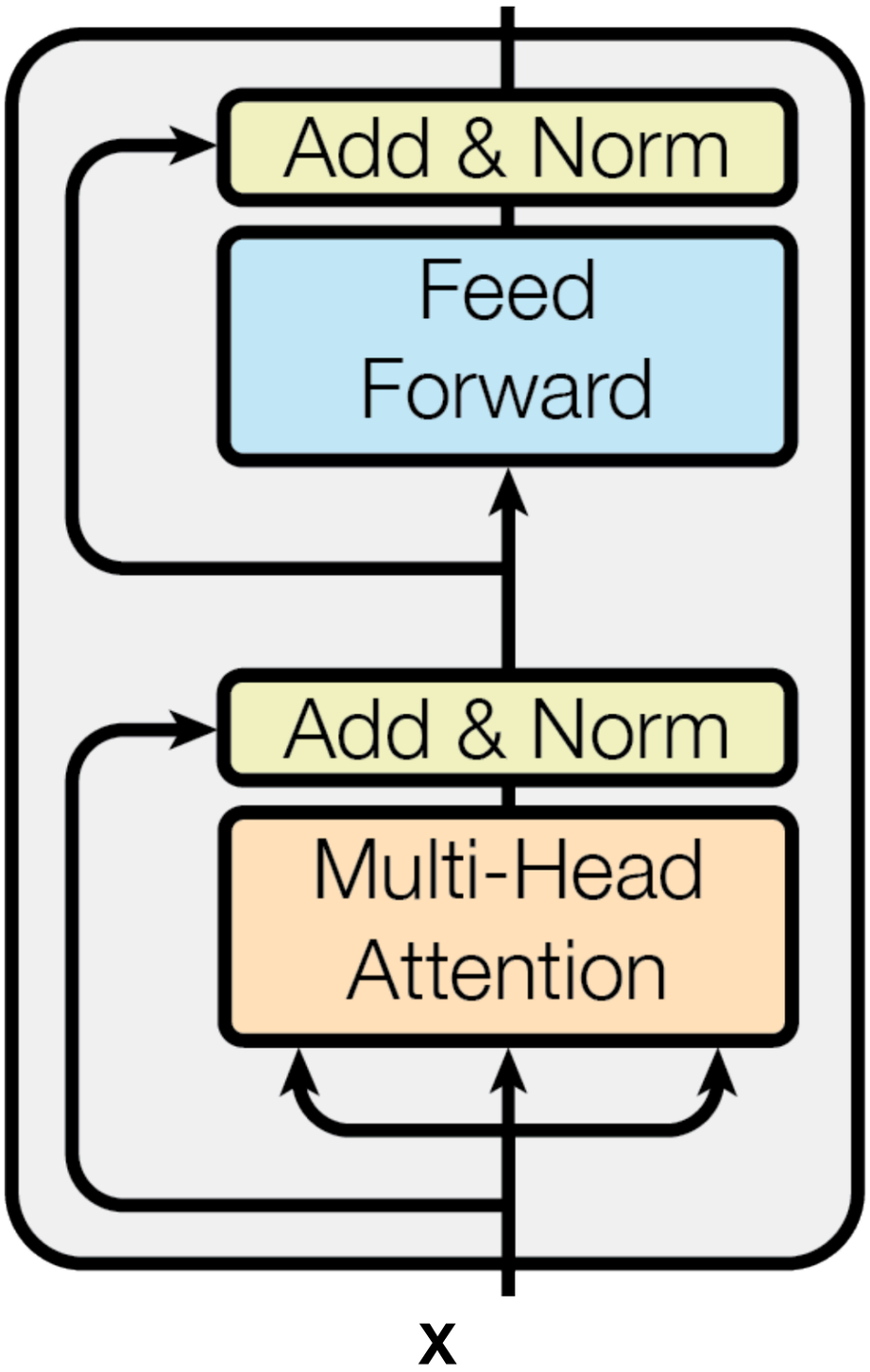


Multi-Head Attention

Attention Heads Visualization

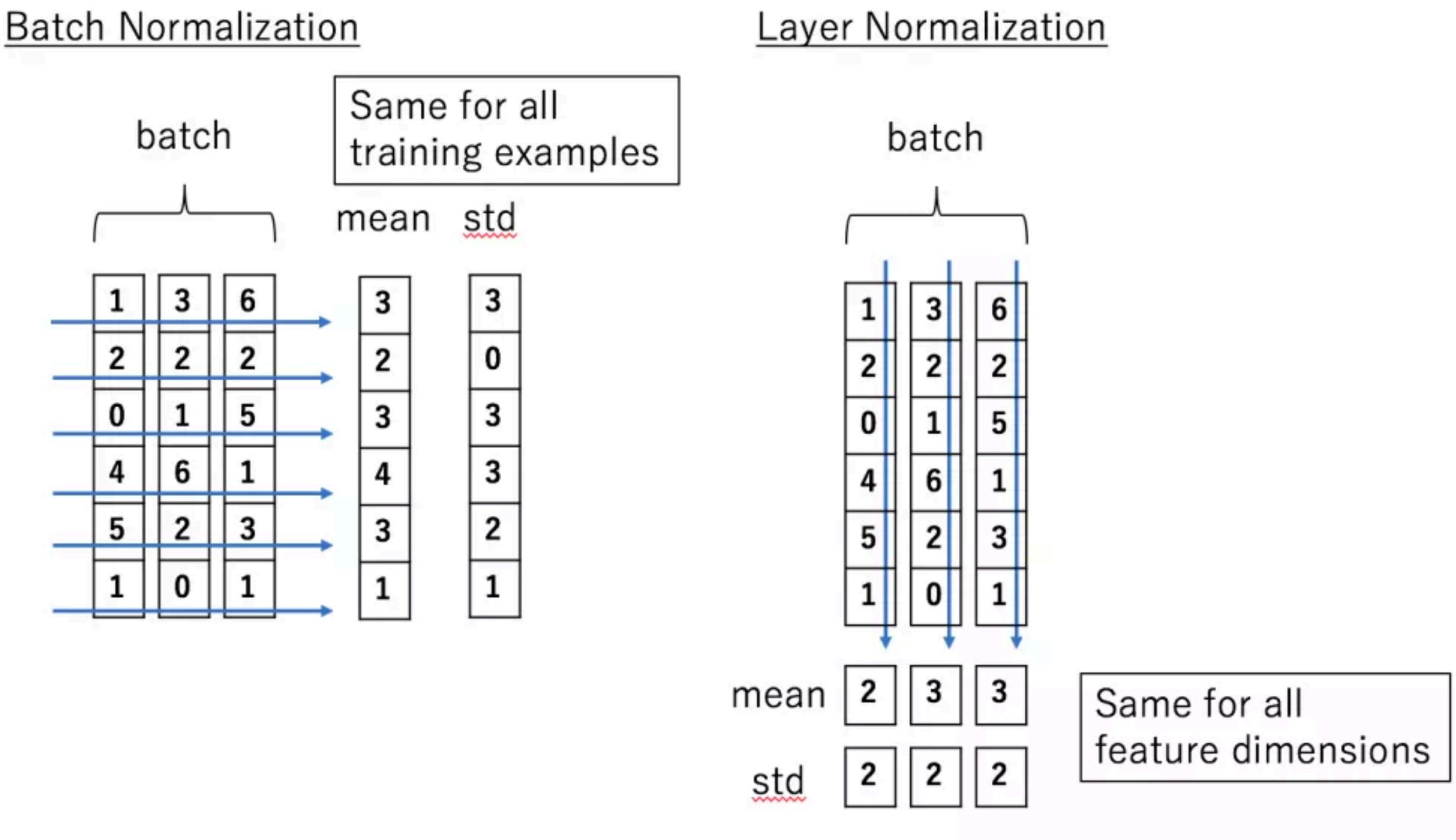


Transformer Encoder

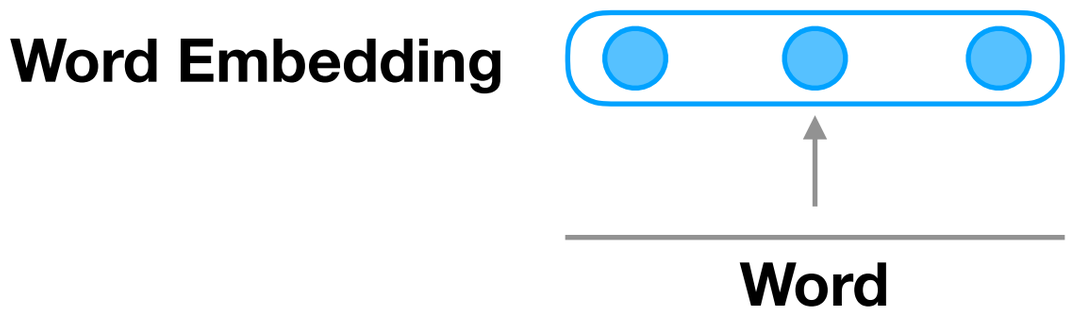


- Add & Norm = LayerNorm(x + sublayer(x))
- LayerNorm changes input to have mean 0 and variance 1 for faster convergence
- Residual connections
- Dropout with small probability (0.1)

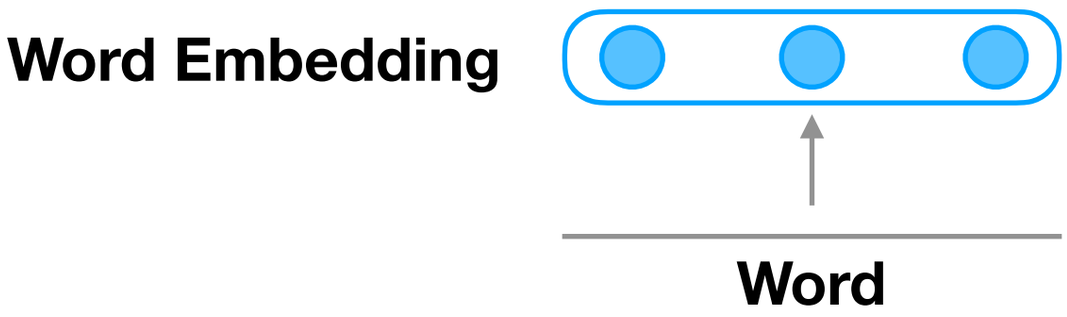
Transformer Encoder



Feed Forward



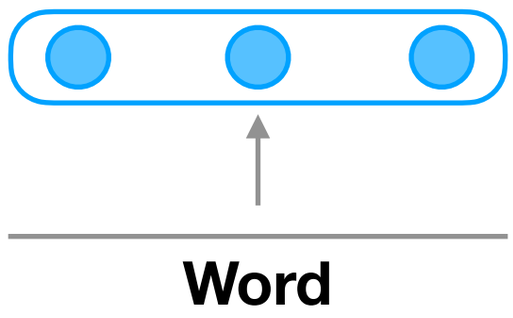
Feed Forward



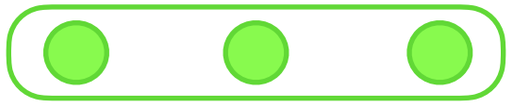
Feed Forward

Usually 4X of previous dimension

Word Embedding

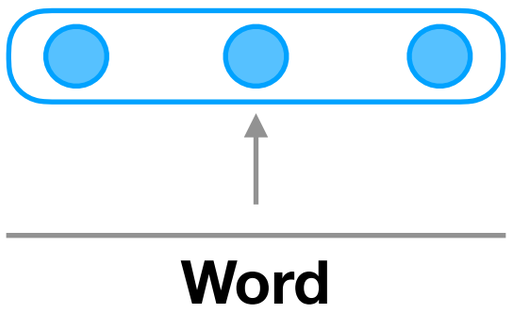


Feed Forward



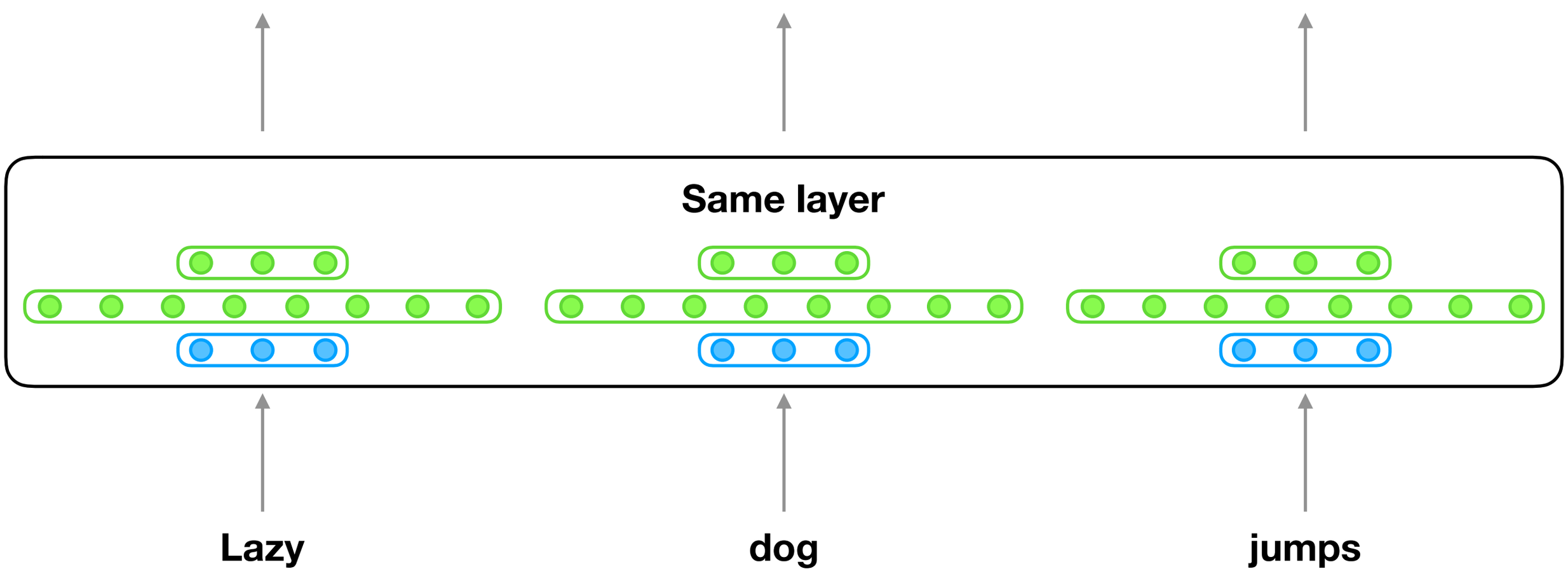
Usually 4X of previous dimension

Word Embedding

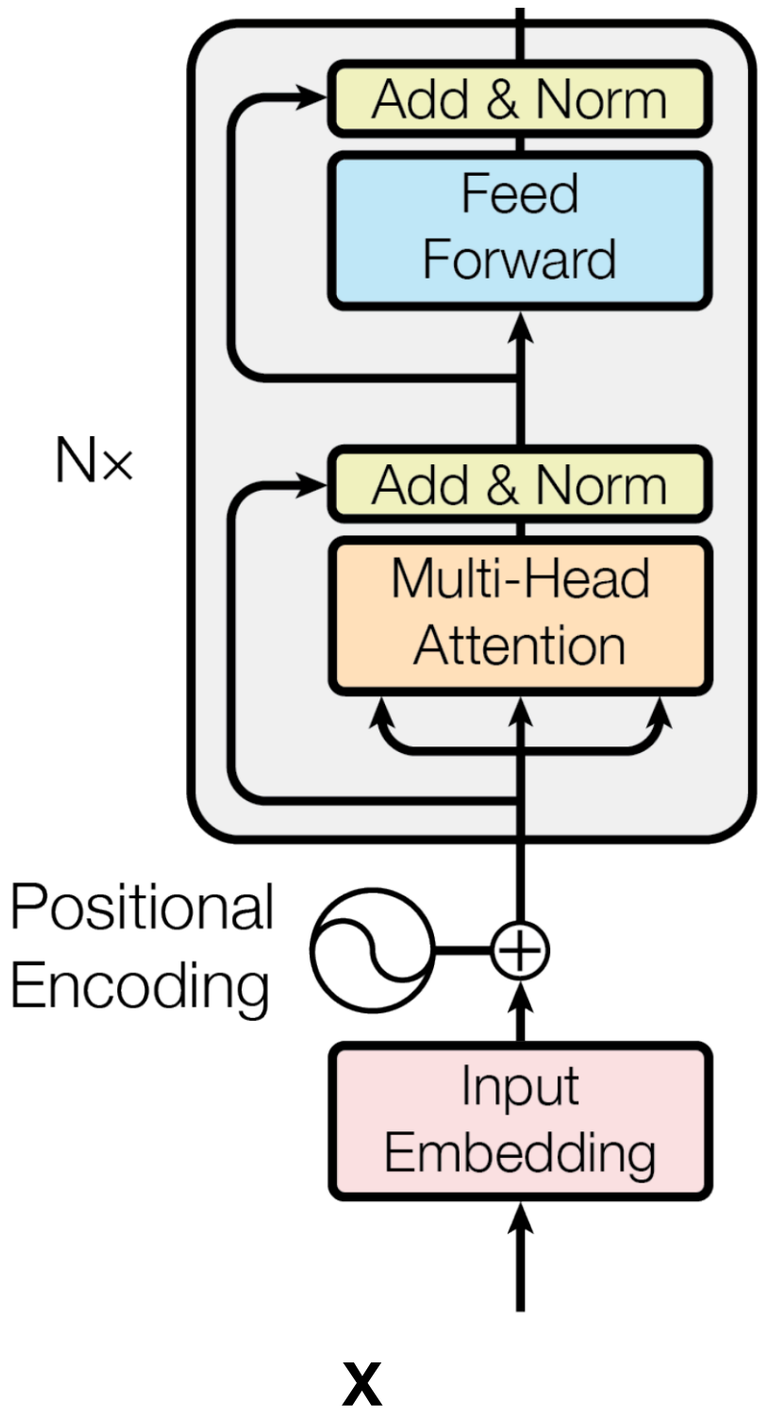


Word

Position-Wise Feed Forward

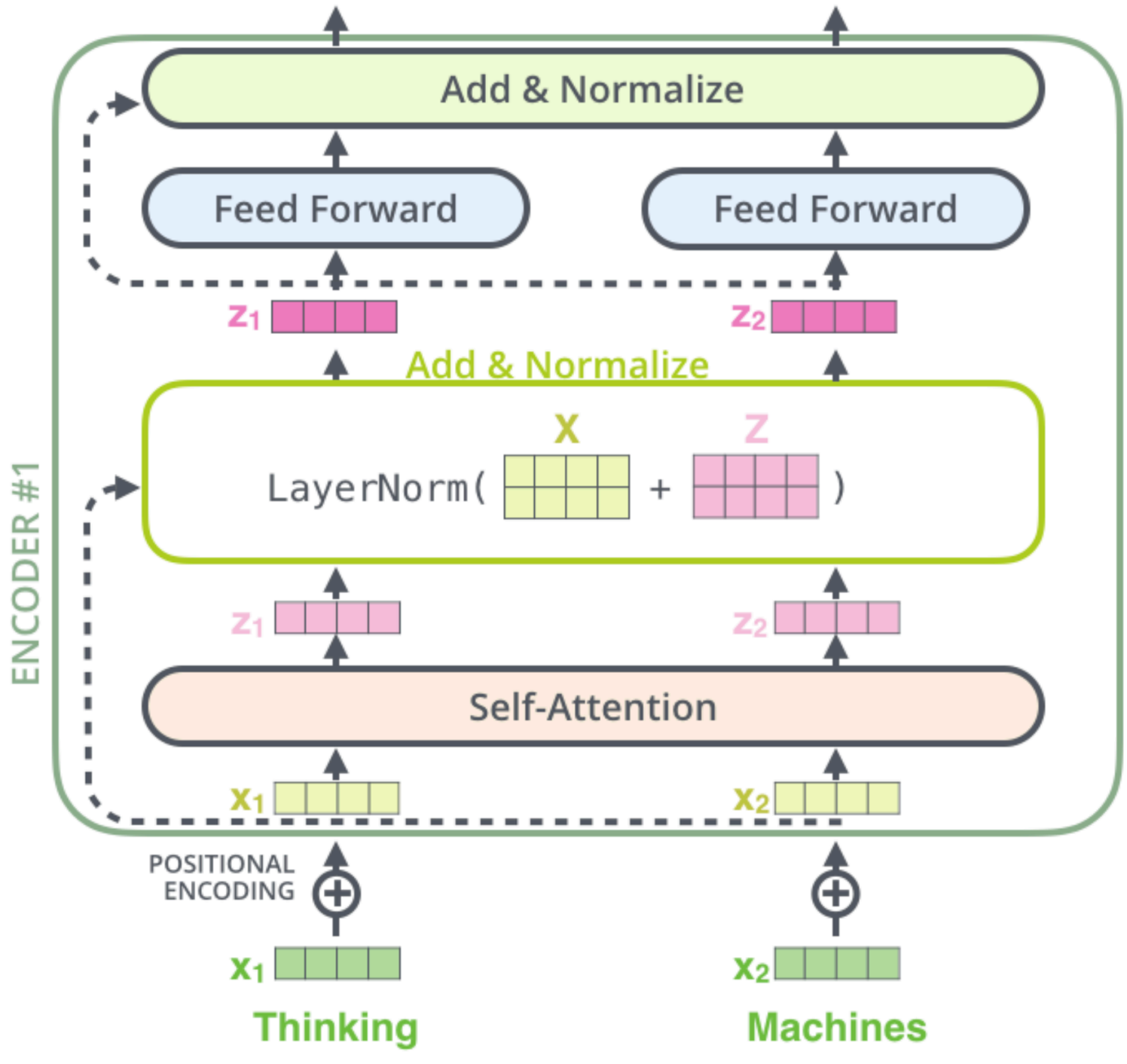


Transformer Encoder

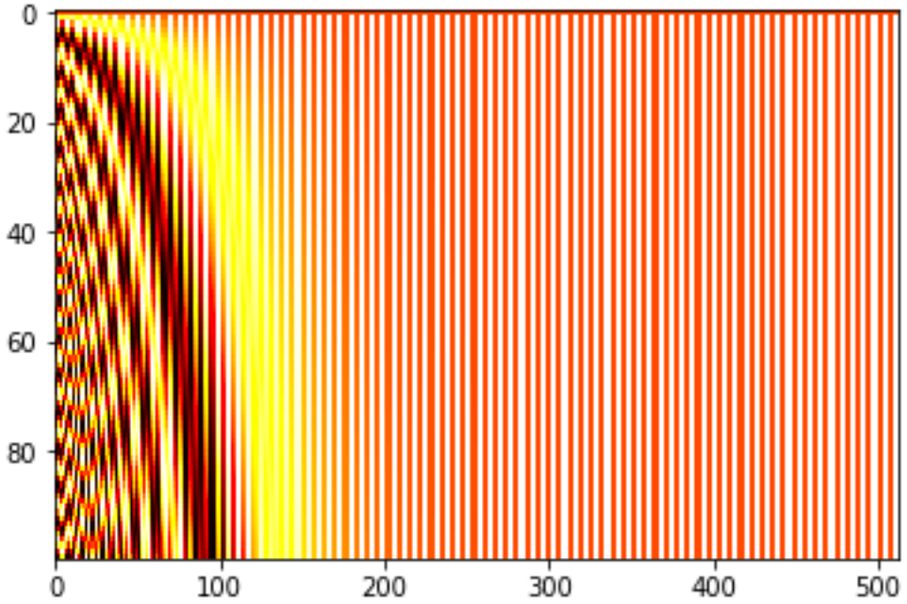


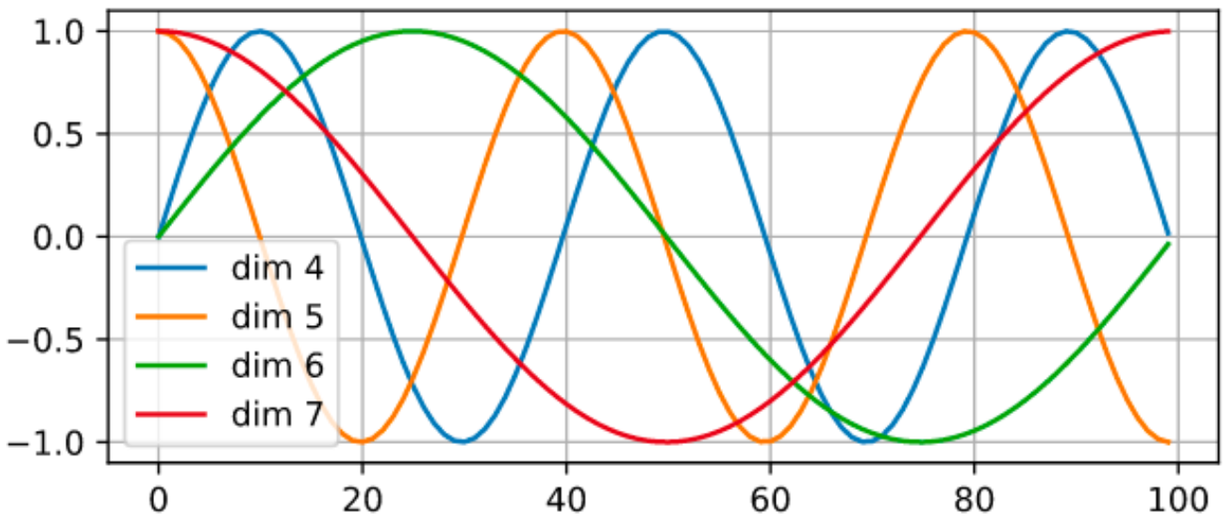
- Nx — just stacked layers for more capacity
- Positional Encoding give information about position of word in text (can be heuristic or learnable)
- Input Embedding just common lookup table

Transformer Encoder



Positional Encoding



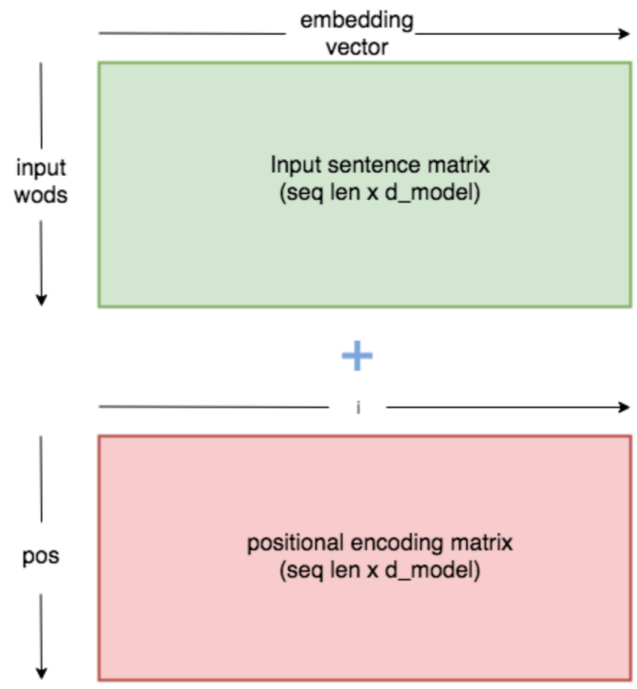


$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

i — index in embedding

- We can calculate positional encoding for every length
- It’s hypothesis and heuristic
- Same results compare to learnable positional embeddings



Positional Embedding

```
positional_embedding = torch.nn.Embedding(num_embeddings=x.shape[0], embedding_dim=x.shape[-1])
```

```
positional_embedding
```

```
Embedding(8, 64)
```

```
positions = torch.arange(start=0, end=x.shape[0])
positions
```

```
tensor([0, 1, 2, 3, 4, 5, 6, 7])
```

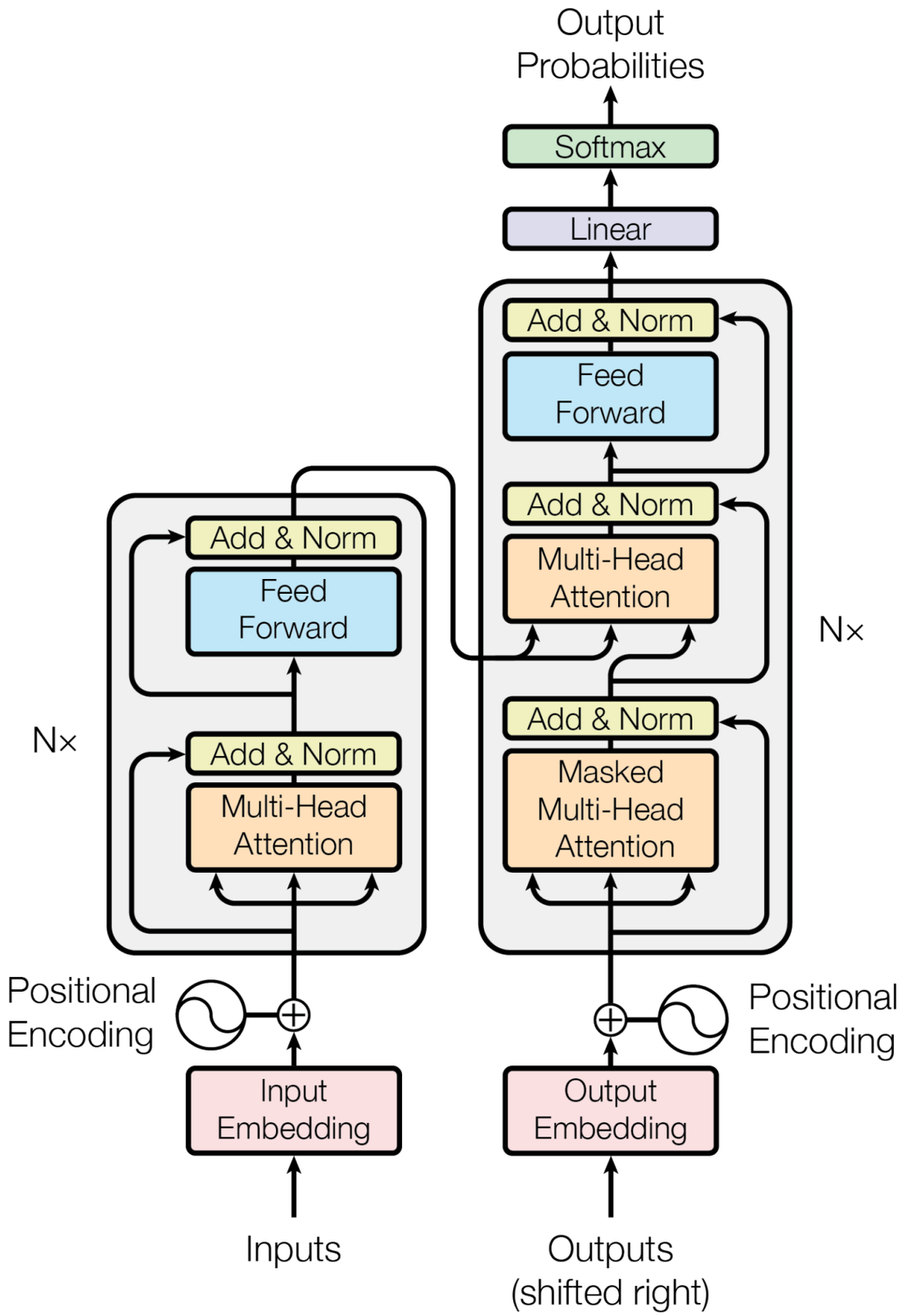
```
pos_emb = positional_embedding(positions)
```

```
x.shape == pos_emb.shape
```

```
True
```

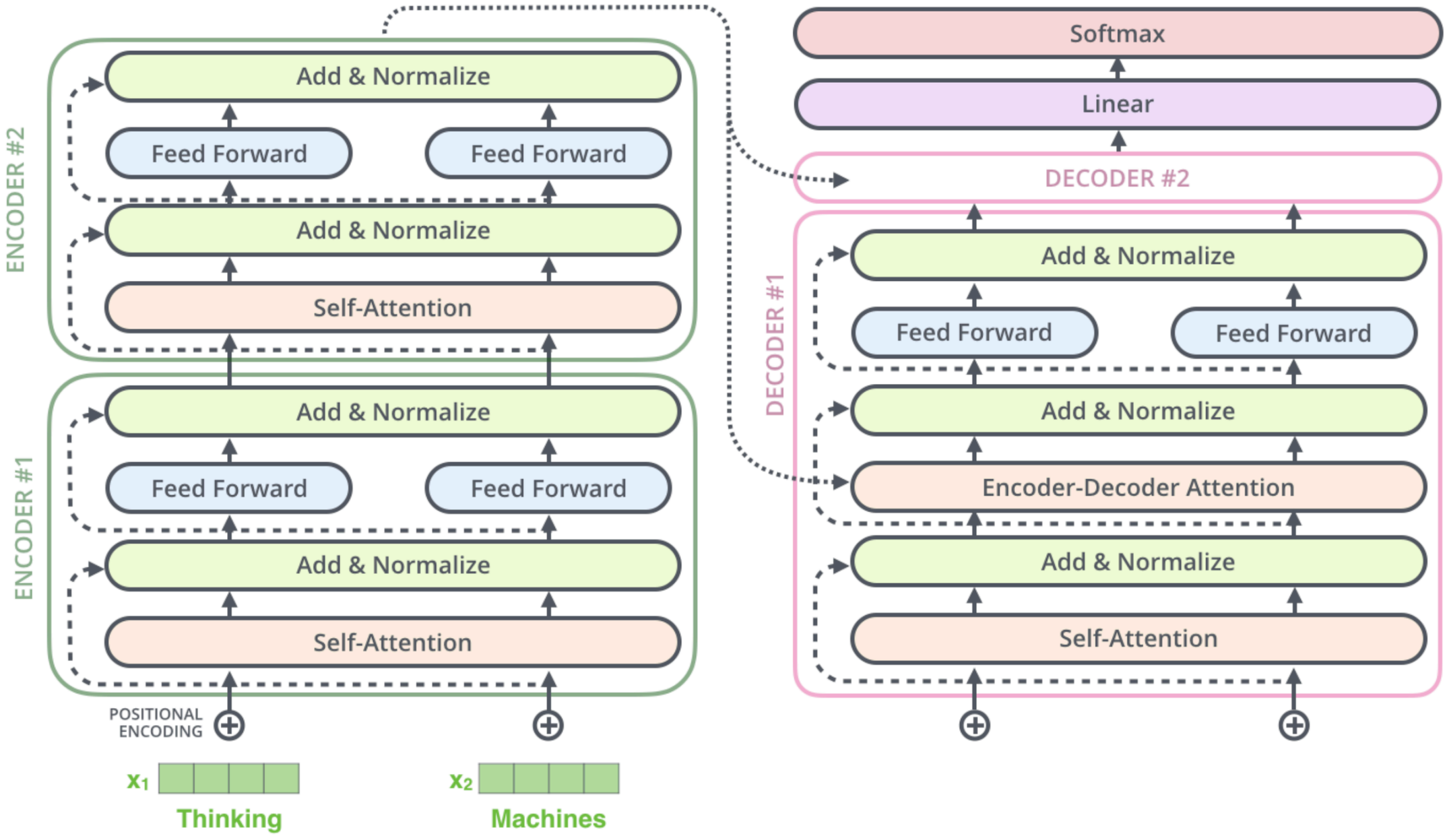
```
x = x + pos_emb
```

Transformer



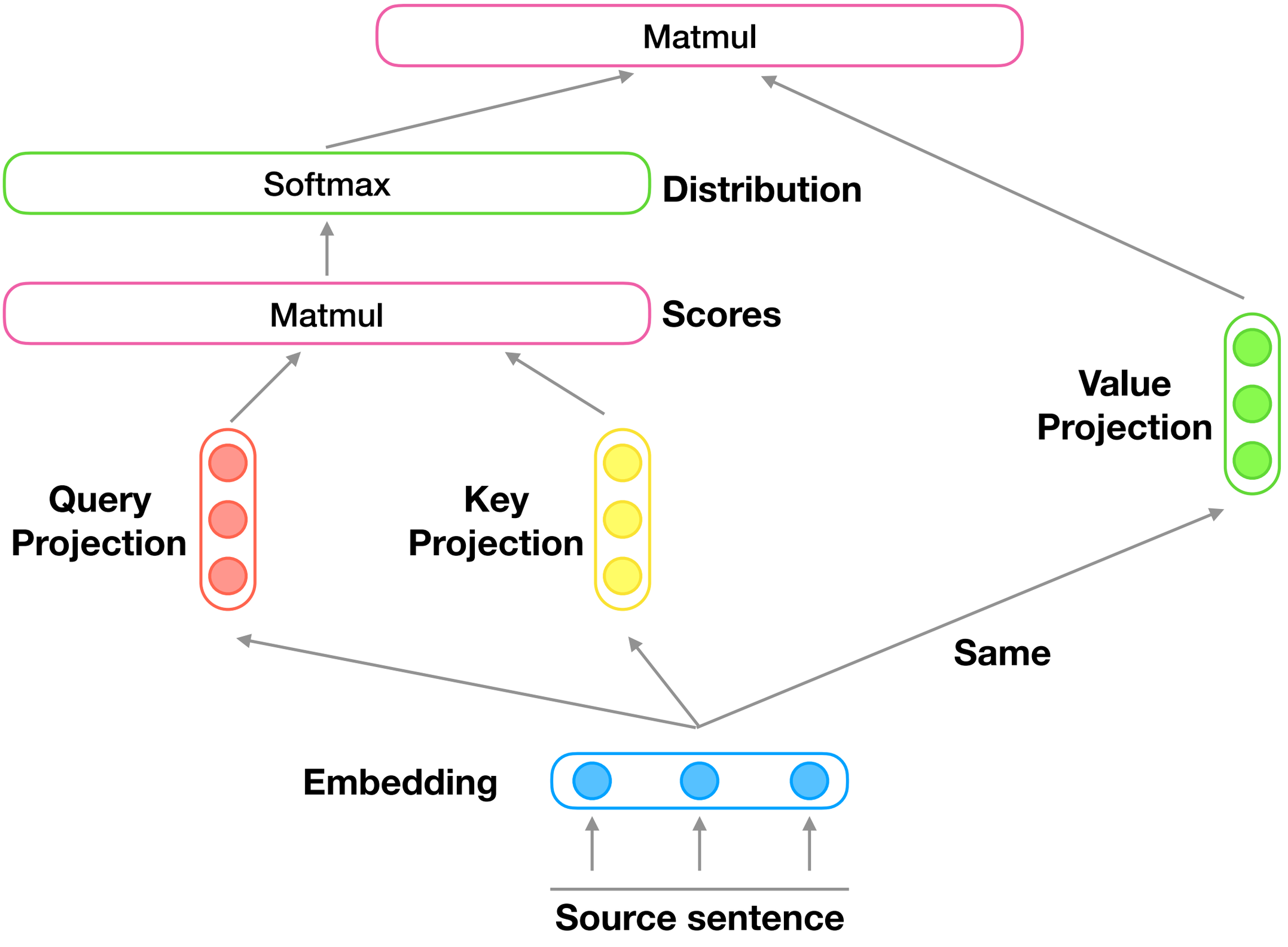
- Masking to ensure that we do not look into the future
- Masks setting to ***-inf*** then we apply softmax and this word becomes zero
- Linear and softmax is just word prediction

Transformer

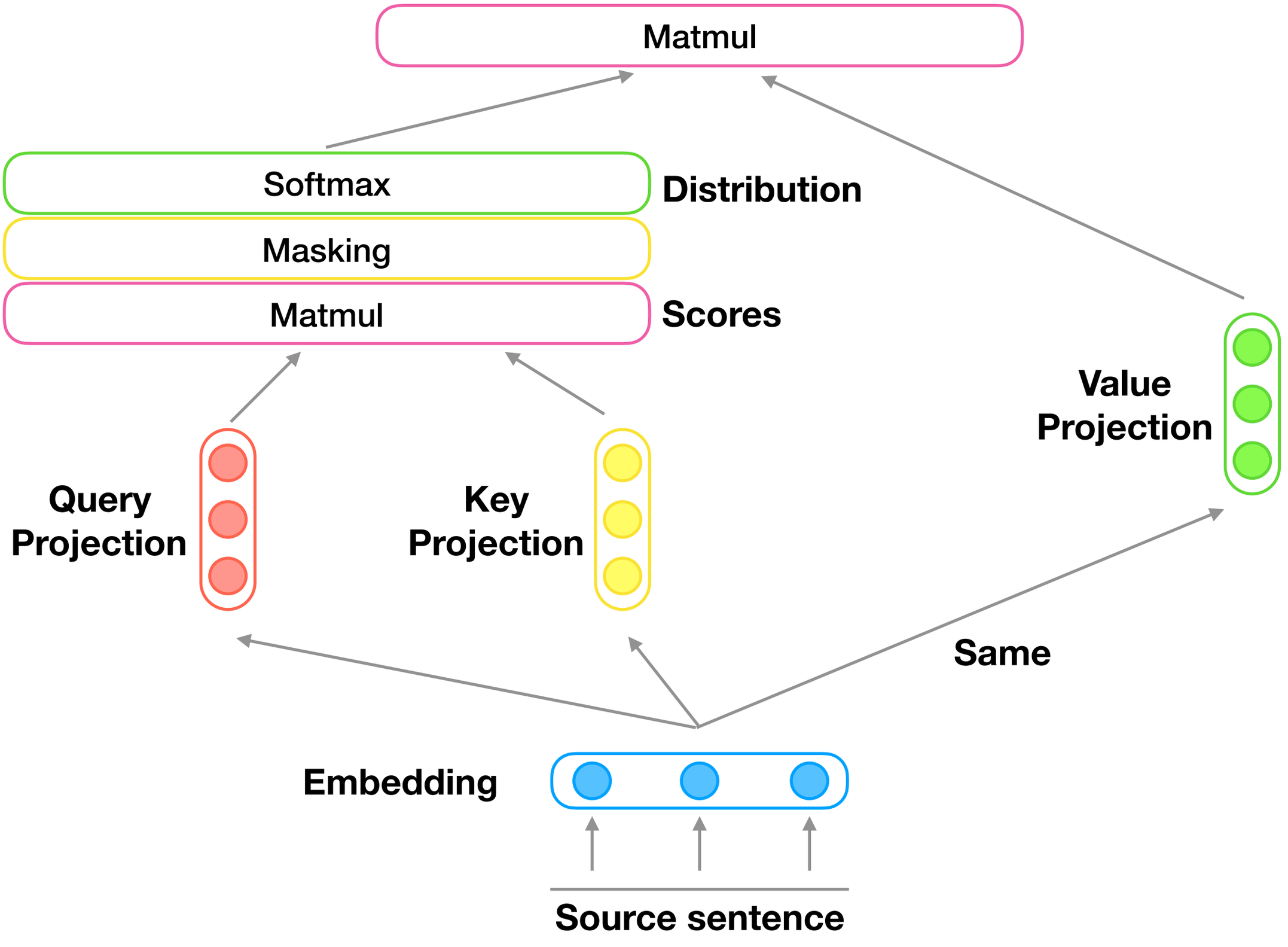


Masking

Self-Attention



Masked Self-Attention



Masking

Source text	I	am	space	invader

Masking

Source text

I am space invader

Attention Scores

0.11	0.04	0.05	0.3
0.19	0.53	0.42	0.37
0.81	0.21	0.05	0.09
0.51	0.43	0.12	0.03

Masking

Source text

I am space invader

Attention Scores

0.11	0.04	0.05	0.3
0.19	0.53	0.42	0.37
0.81	0.21	0.05	0.09
0.51	0.43	0.12	0.03

Masking

Future

Masked Attention Scores

0.11	-inf	-inf	-inf
0.19	0.53	-inf	-inf
0.81	0.21	0.05	-inf
0.51	0.43	0.12	0.03

Masking

Source text

I am space invader

Attention Scores

0.11	0.04	0.05	0.3
0.19	0.53	0.42	0.37
0.81	0.21	0.05	0.09
0.51	0.43	0.12	0.03

Masking

Future

Masked Attention Scores

Time

0.11	-inf	-inf	-inf
0.19	0.53	-inf	-inf
0.81	0.21	0.05	-inf
0.51	0.43	0.12	0.03

Masking

Source text

I am space invader

Attention Scores

0.11	0.04	0.05	0.3
0.19	0.53	0.42	0.37
0.81	0.21	0.05	0.09
0.51	0.43	0.12	0.03

Masking

→

Future

Masked Attention Scores

Time →

0.11	-inf	-inf	-inf
0.19	0.53	-inf	-inf
0.81	0.21	0.05	-inf
0.51	0.43	0.12	0.03

Masking

Source text I am space invader

Attention Scores

0.11	0.04	0.05	0.3
0.19	0.53	0.42	0.37
0.81	0.21	0.05	0.09
0.51	0.43	0.12	0.03

Masking

Future

Masked Attention Scores

Time

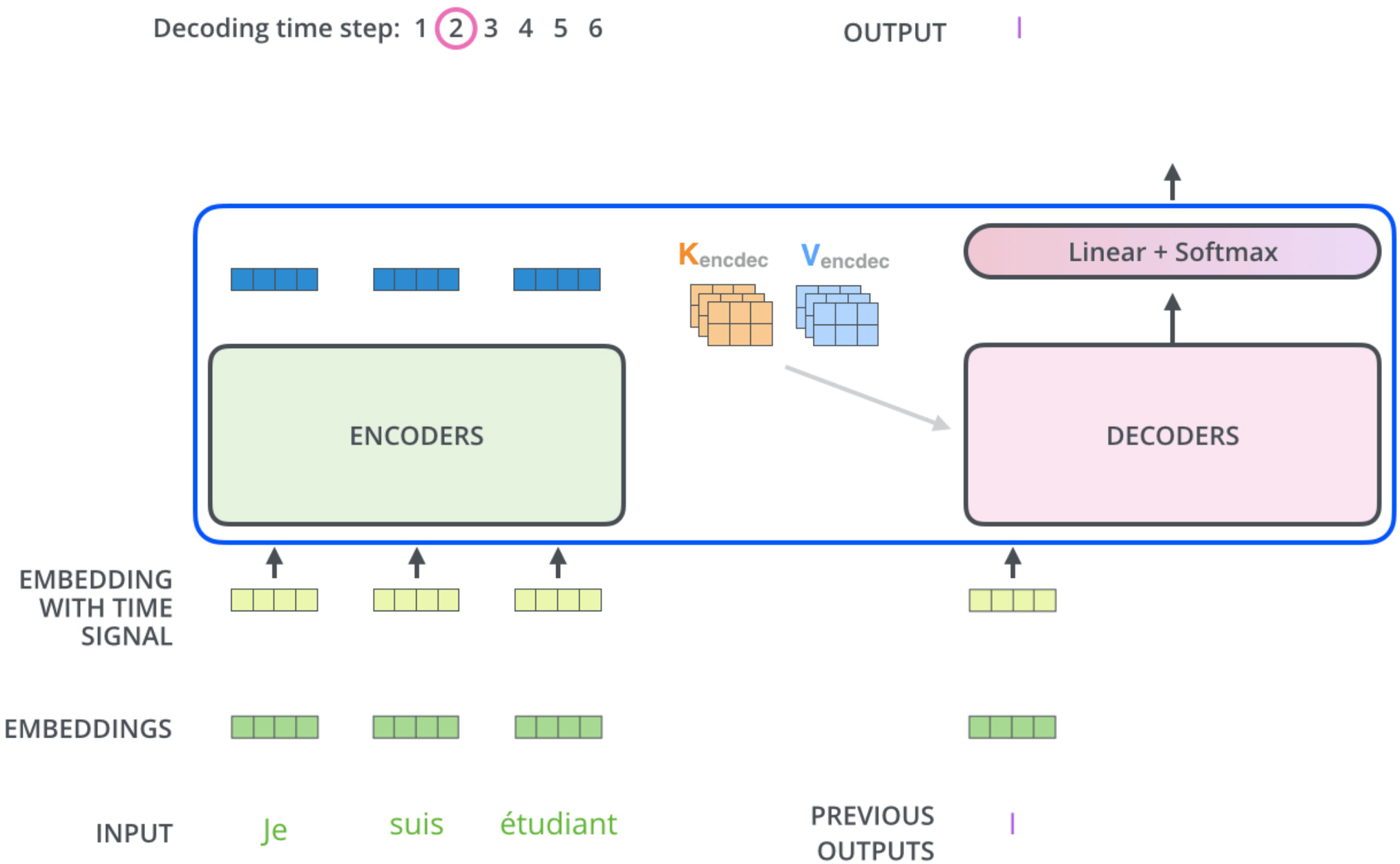
0.11	-inf	-inf	-inf
0.19	0.53	-inf	-inf
0.81	0.21	0.05	-inf
0.51	0.43	0.12	0.03

Softmax

Attention Distribution

1	0	0	0
0.48	0.52	0	0
0.45	0.21	0.34	0
0.25	0.16	0.33	0.26

Decoding


Decoding time step: 1 2 3 4 5 6
OUTPUT
I
Linear + Softmax
K encdec
V encdec
ENCODERS
DECODERS
EMBEDDING WITH TIME SIGNAL
EMBEDDINGS
INPUT
Je
suis
étudiant
PREVIOUS OUTPUTS
I

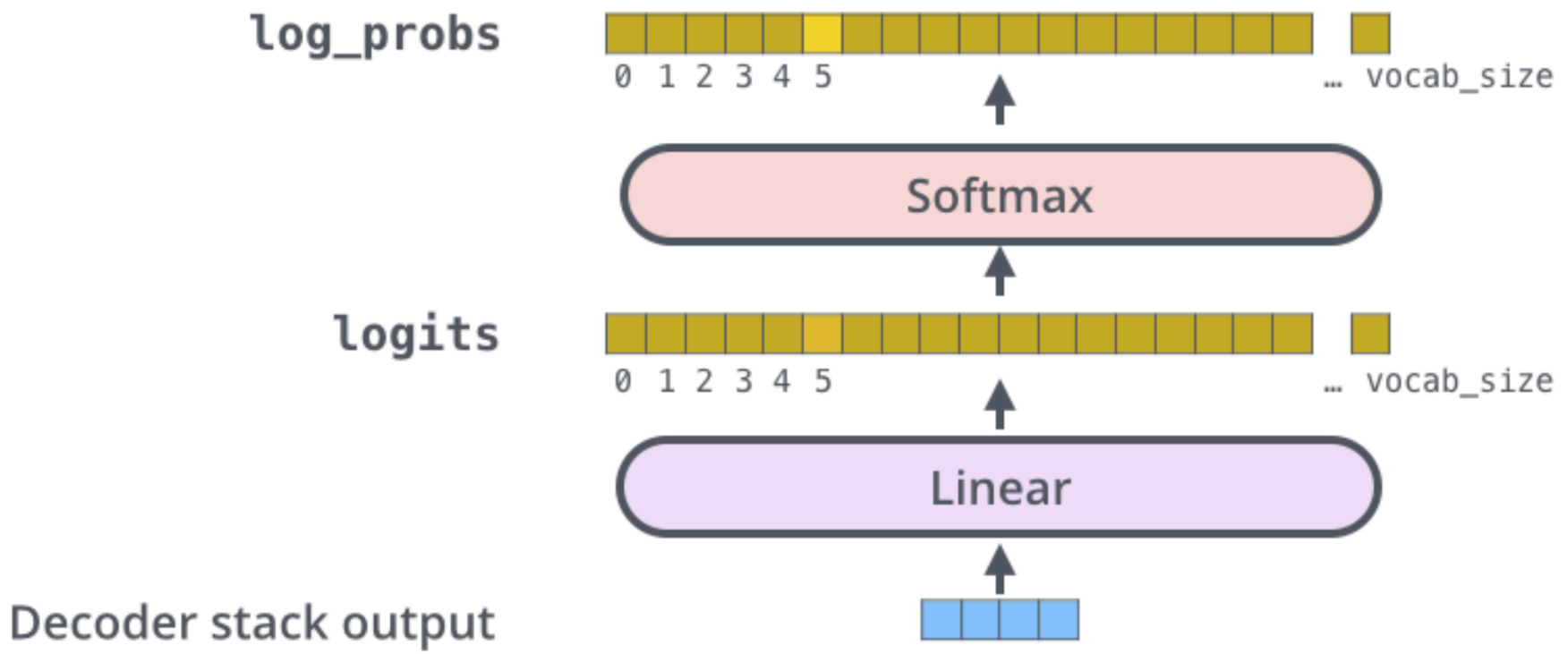
Decoding

Which word in our vocabulary is associated with this index?

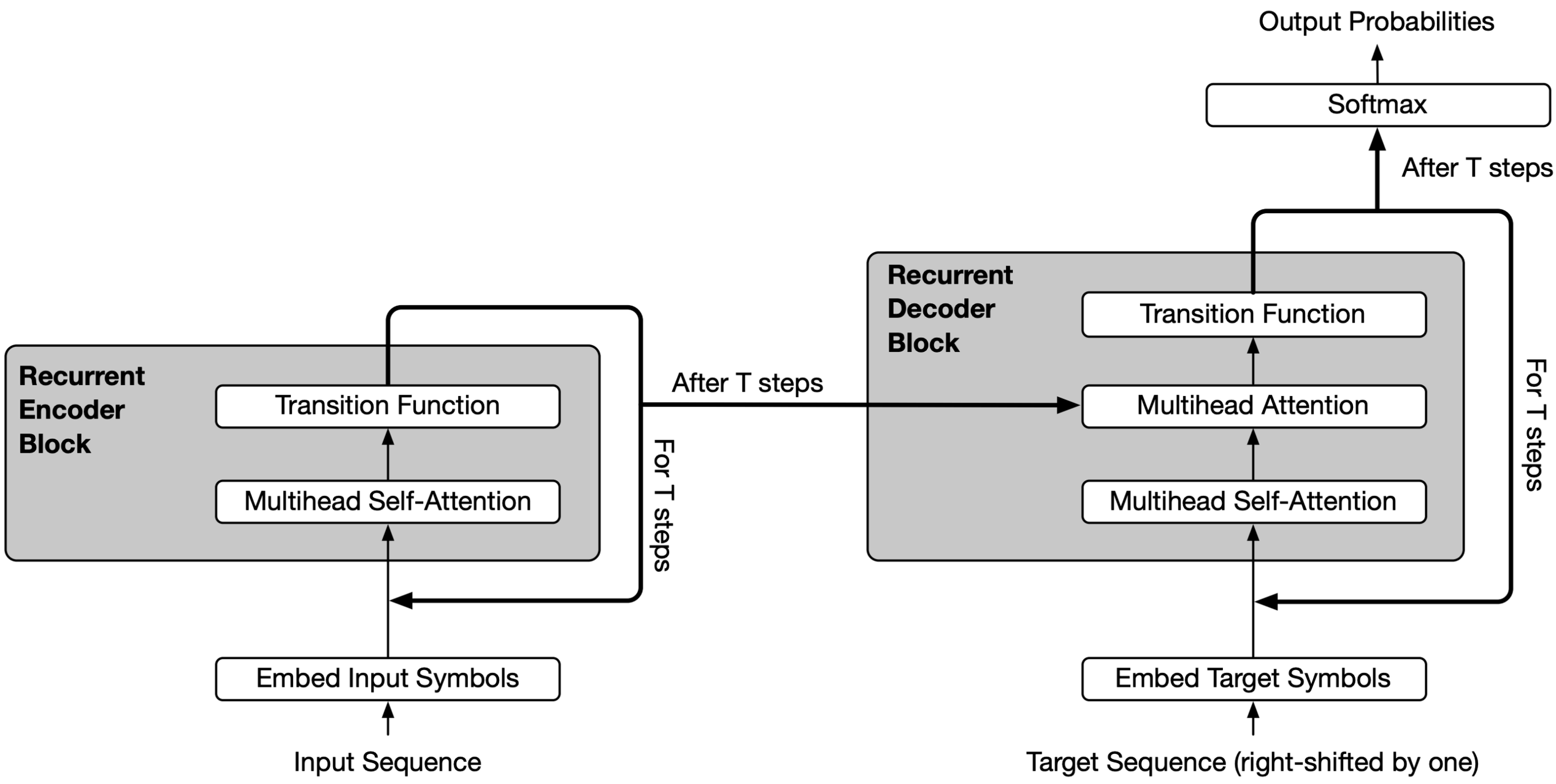
am

Get the index of the cell with the highest value (`argmax`)

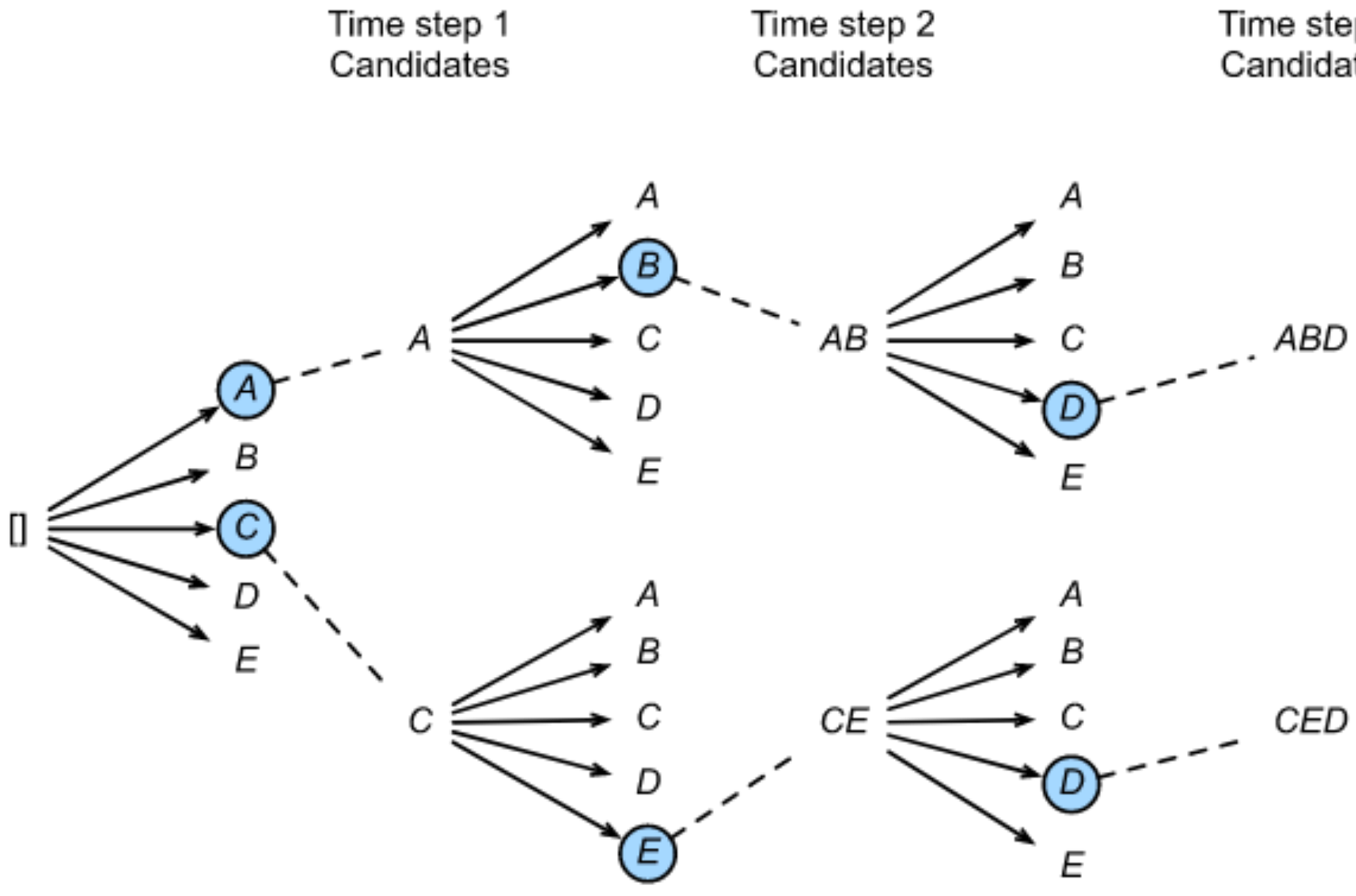
5



Universal Transformer



Beam Search


Time step 1
Candidates
Time step 2
Candidates
Time step 3
Candidates
[]
A
B
C
D
E
A
B
C
D
E
AB
CE
ABD
CED

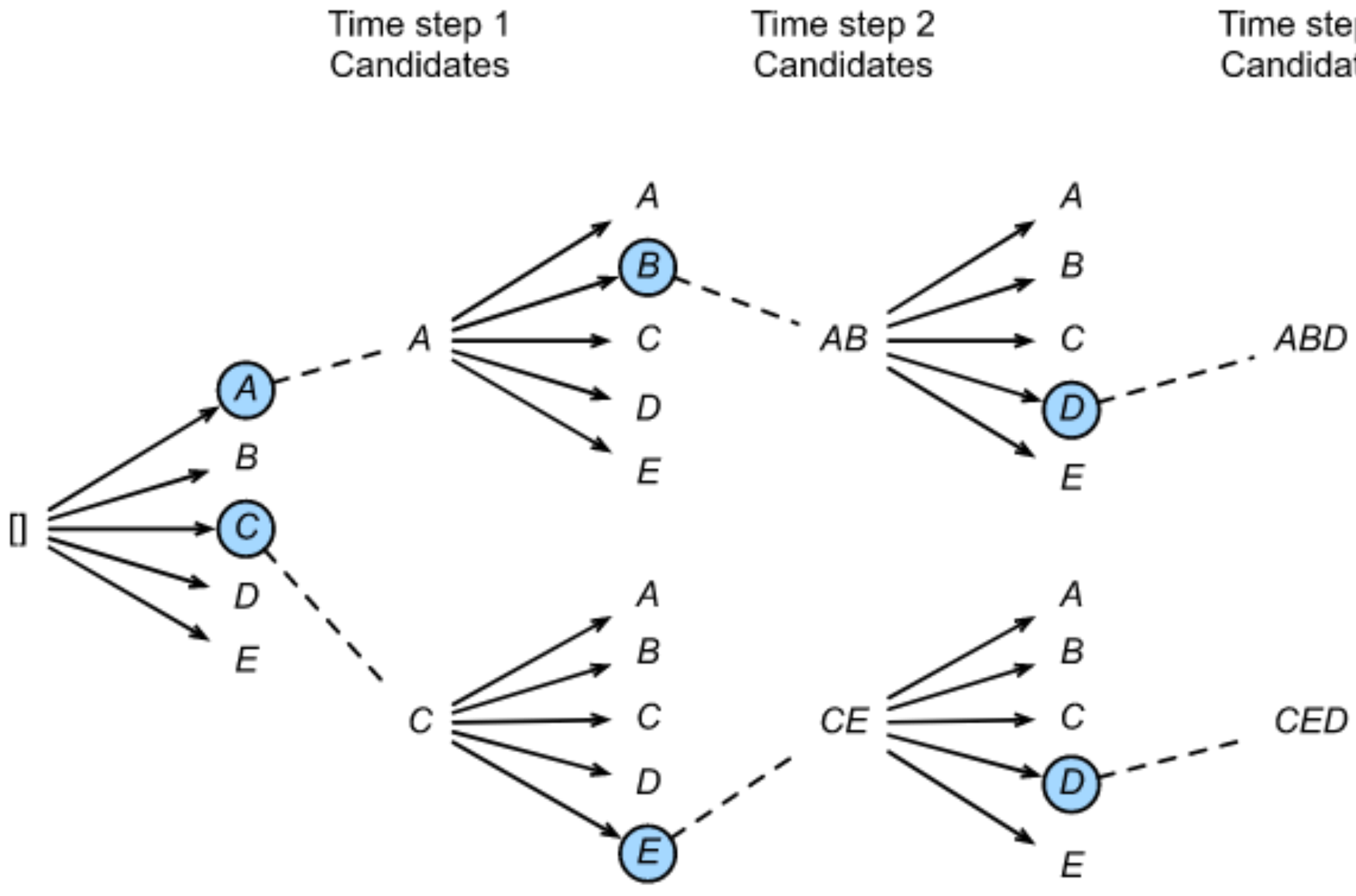
Beam Search

$$p(\mathbf{x}) = \prod_i p(x|x_{<i}) = p(x_0)p(x_1|x_0)p(x_2|x_0, x_1)...$$

$$\arg\max_y \prod_{t=1}^{T_y} P(y^{<t>} | x, y^{<1>}, \ldots, y^{<t-1>})$$

$$\arg\max_y \sum_{y=1}^{T_y} \log P(y^{<t>} | x, y^{<1>}, \ldots, y^{<t-1>})$$

Beam Search


Time step 1
Candidates
Time step 2
Candidates
Time step 3
Candidates
[]
A
B
C
D
E
A
B
C
D
E
AB
CE
ABD
CED