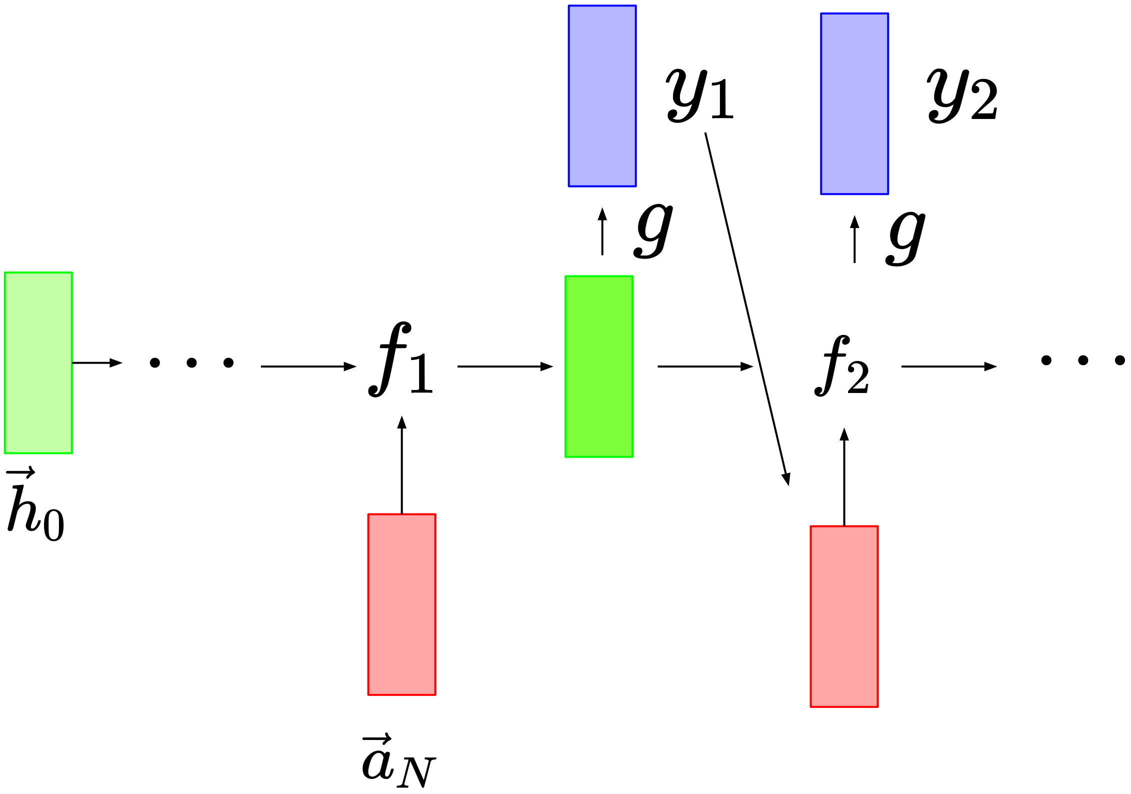
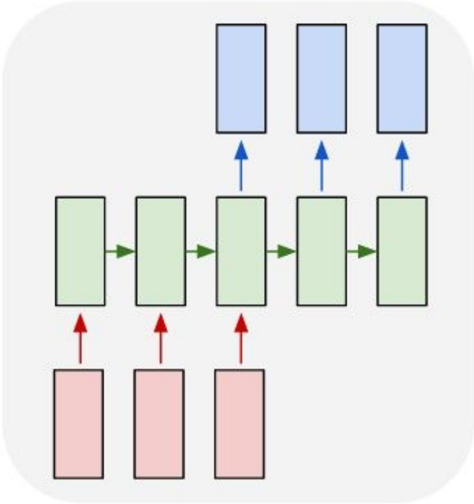
Overview

- Attention
- The Transformer
- BERT

Seq2Seq Models

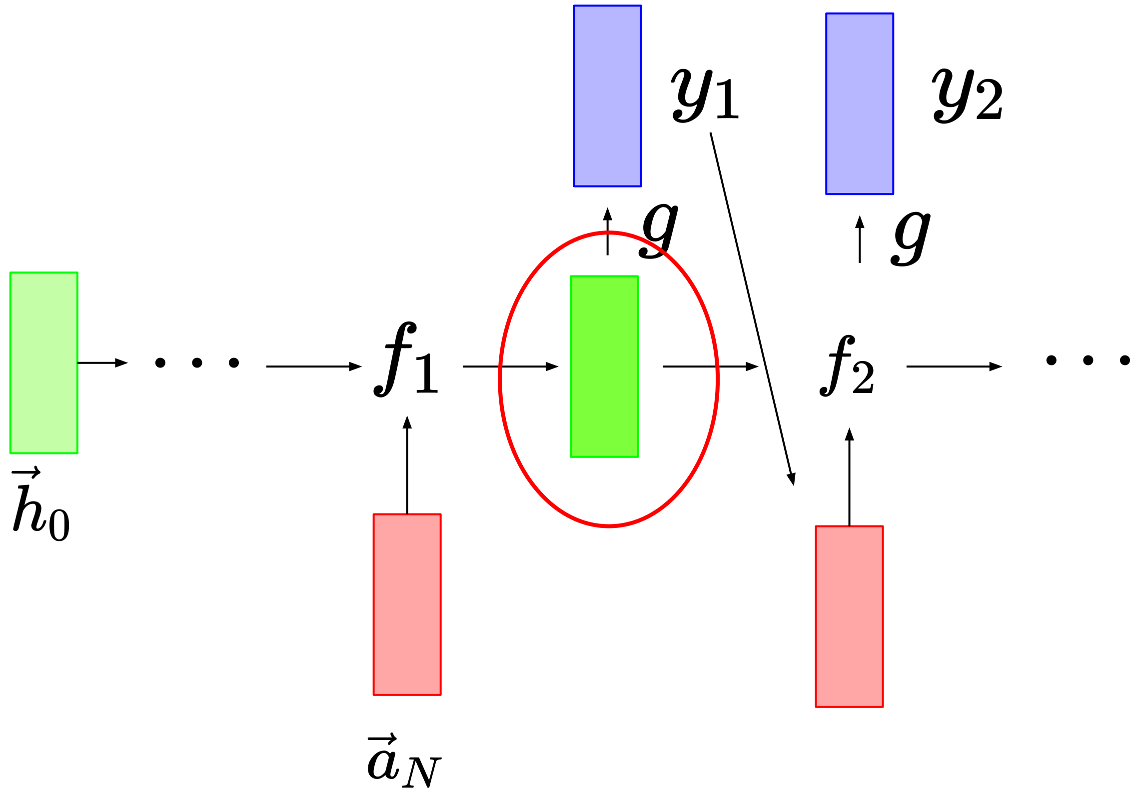


many to many

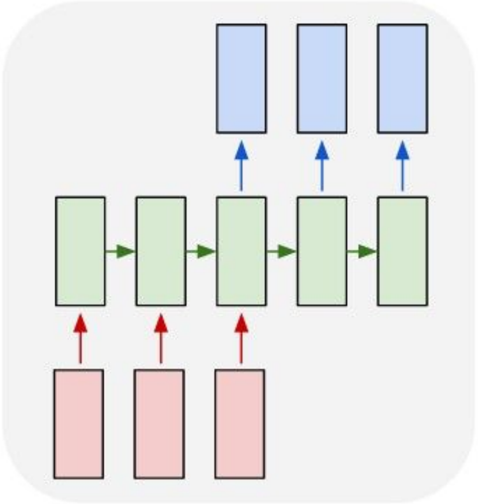


Seq2Seq Models

- Information Bottleneck!



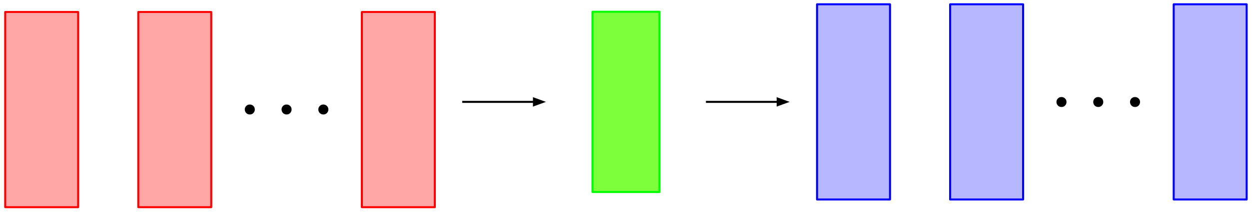
many to many



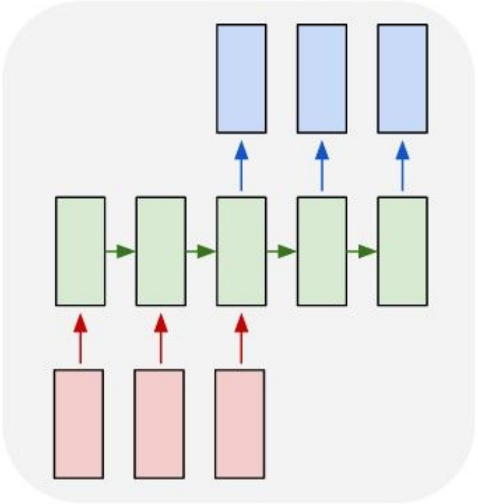
Problems with translation

- Have to encode a full sentence into one vector

Ich muss auf den Markt gehen. → I must go to the Market.



many to many

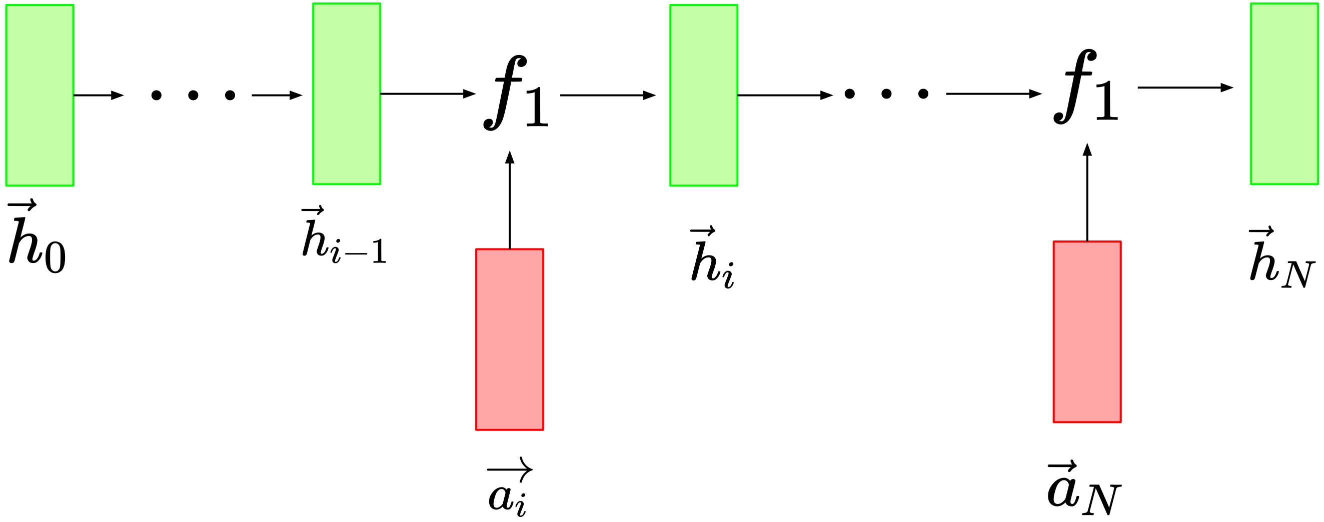


Attention

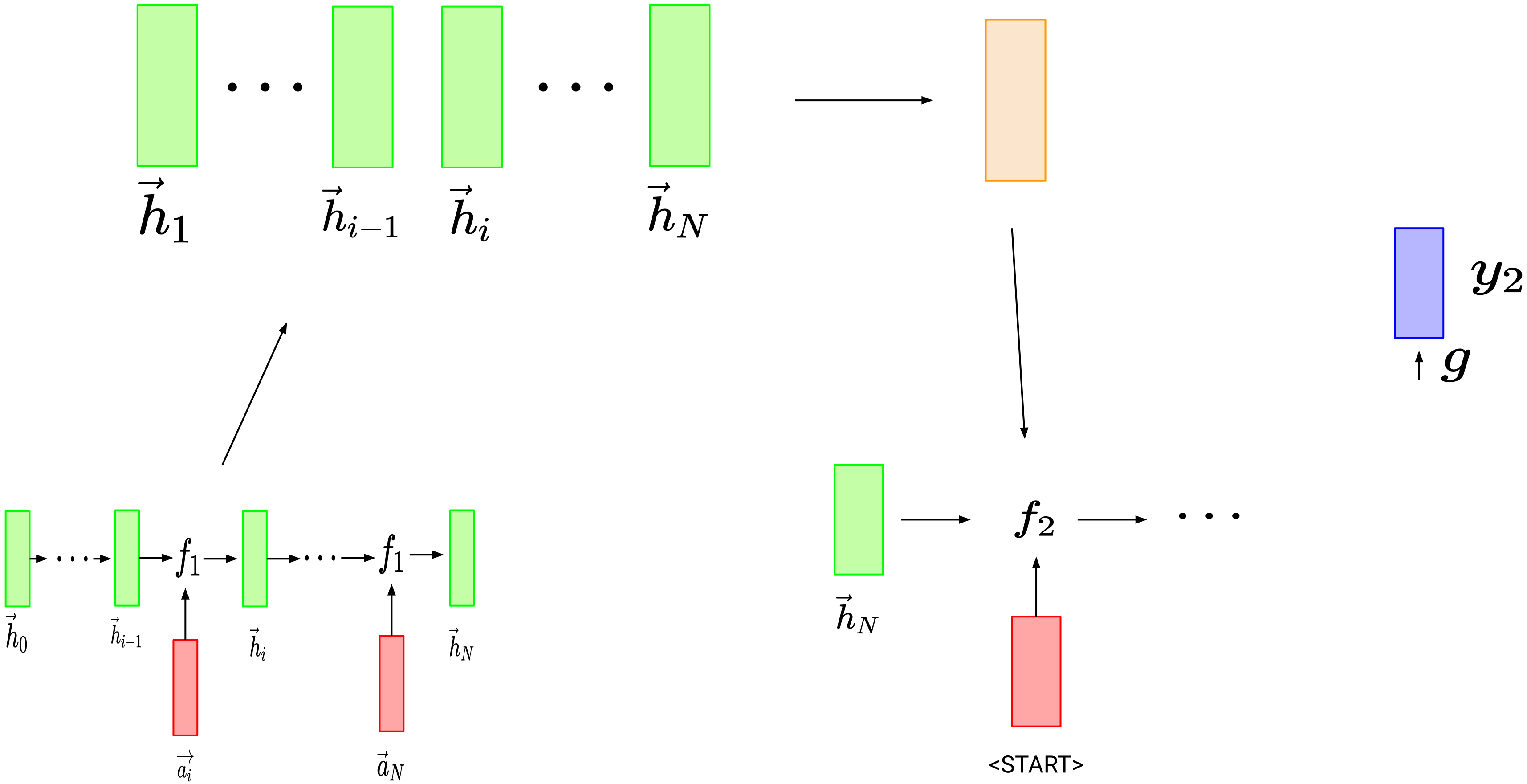
- Use the hidden states generated by the encoder RNN.
- In each decoding step, weight the hidden states (attention mechanism) and use the weighted sum to predict the next element.

Attention

- Use the hidden states generated by the encoder RNN.
- In each decoding step, weight the hidden states (attention mechanism) and use the weighted sum to predict the next element.



Attention

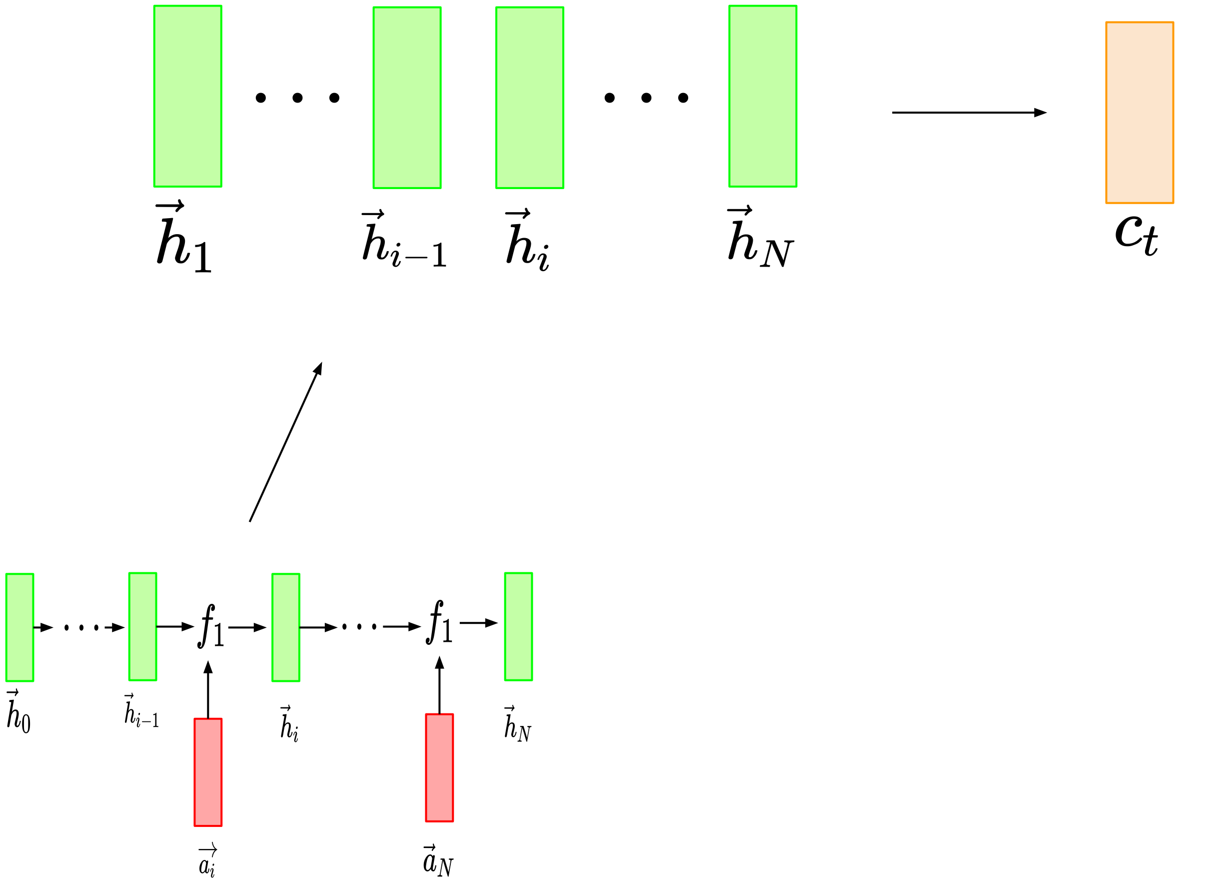


$\vec{h}_1$ $\cdots$ $\vec{h}_{i-1}$ $\vec{h}_i$ $\cdots$ $\vec{h}_N$
y_2
g
$\vec{h}_0$ $\cdots$ $\vec{h}_{i-1}$ f_1 $\vec{h}_i$ $\cdots$ f_1 $\vec{h}_N$
$\vec{a_i}$
$\vec{a}_N$
$\vec{h}_N$ f_2 $\cdots$
<START>

Attention

$$\sum_{i=1}^{N} \alpha_i^t h_i = c_t$$

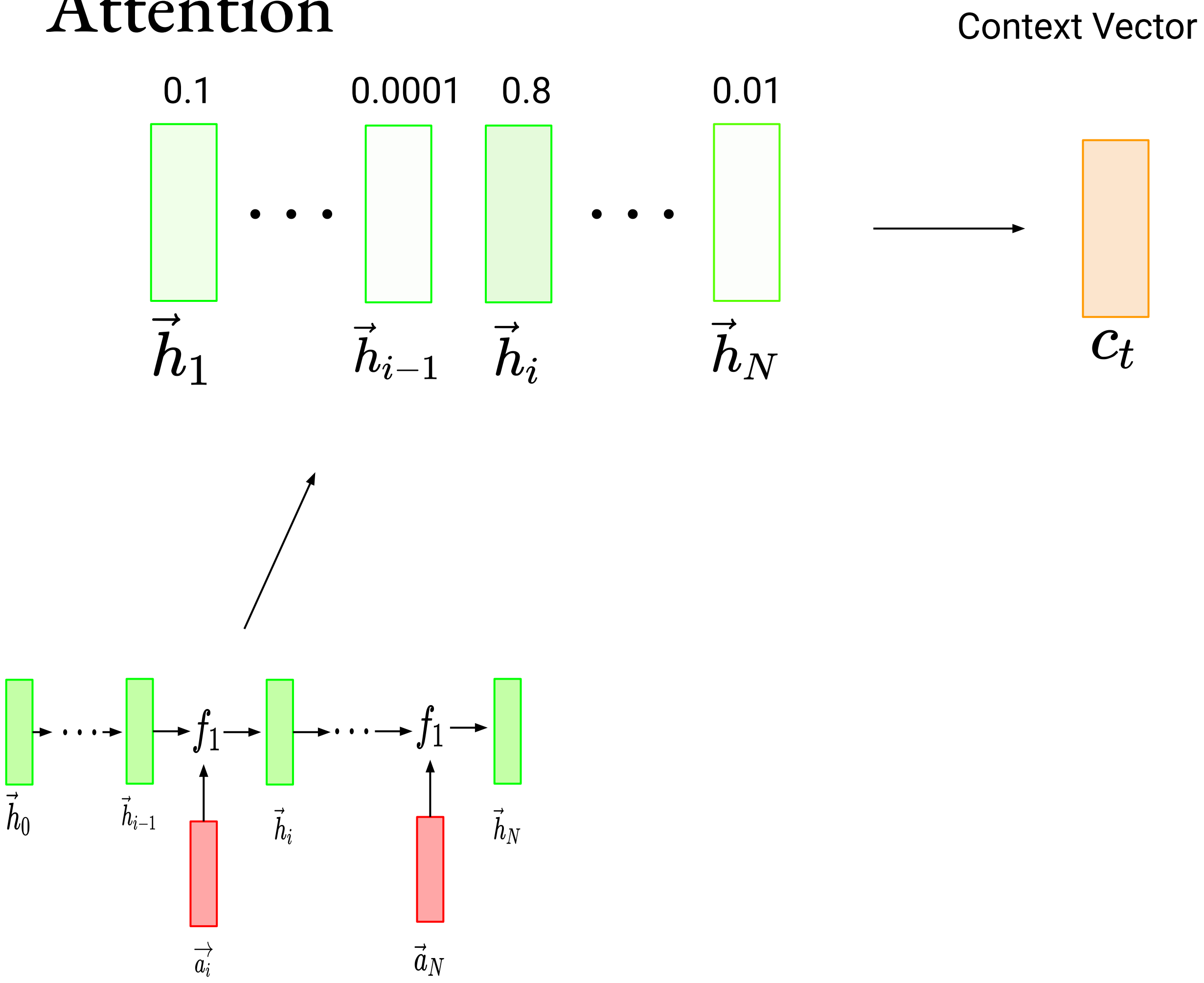
$$\sum_{i=1}^{N} \alpha_i^t = 1$$



Attention

$$\sum_{i=1}^{N} \alpha_i^t h_i = c_t$$

$$\sum_{i=1}^{N} \alpha_i^t = 1$$

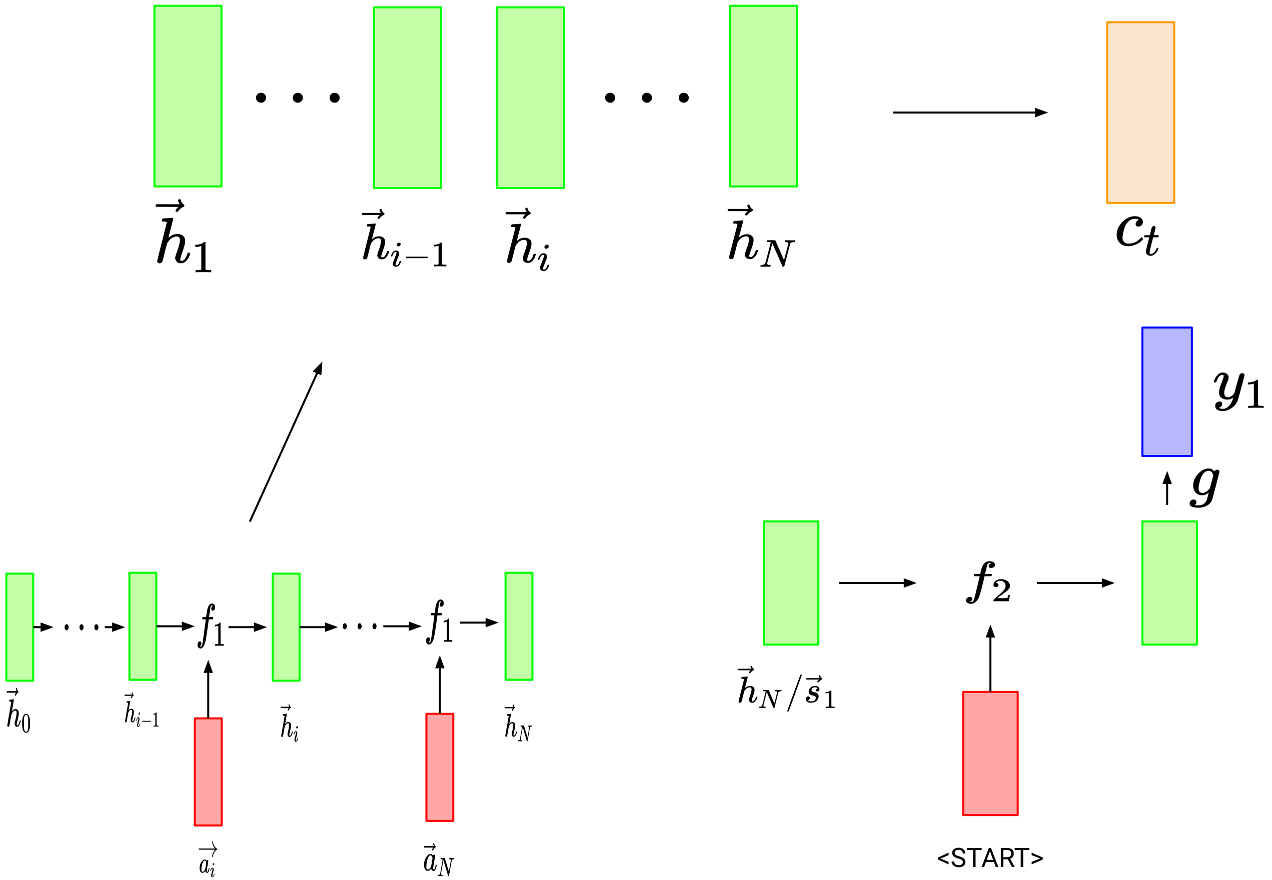


Attention

Context Vector

$$\sum_{i=1}^{N} \alpha_i^t h_i = c_t$$

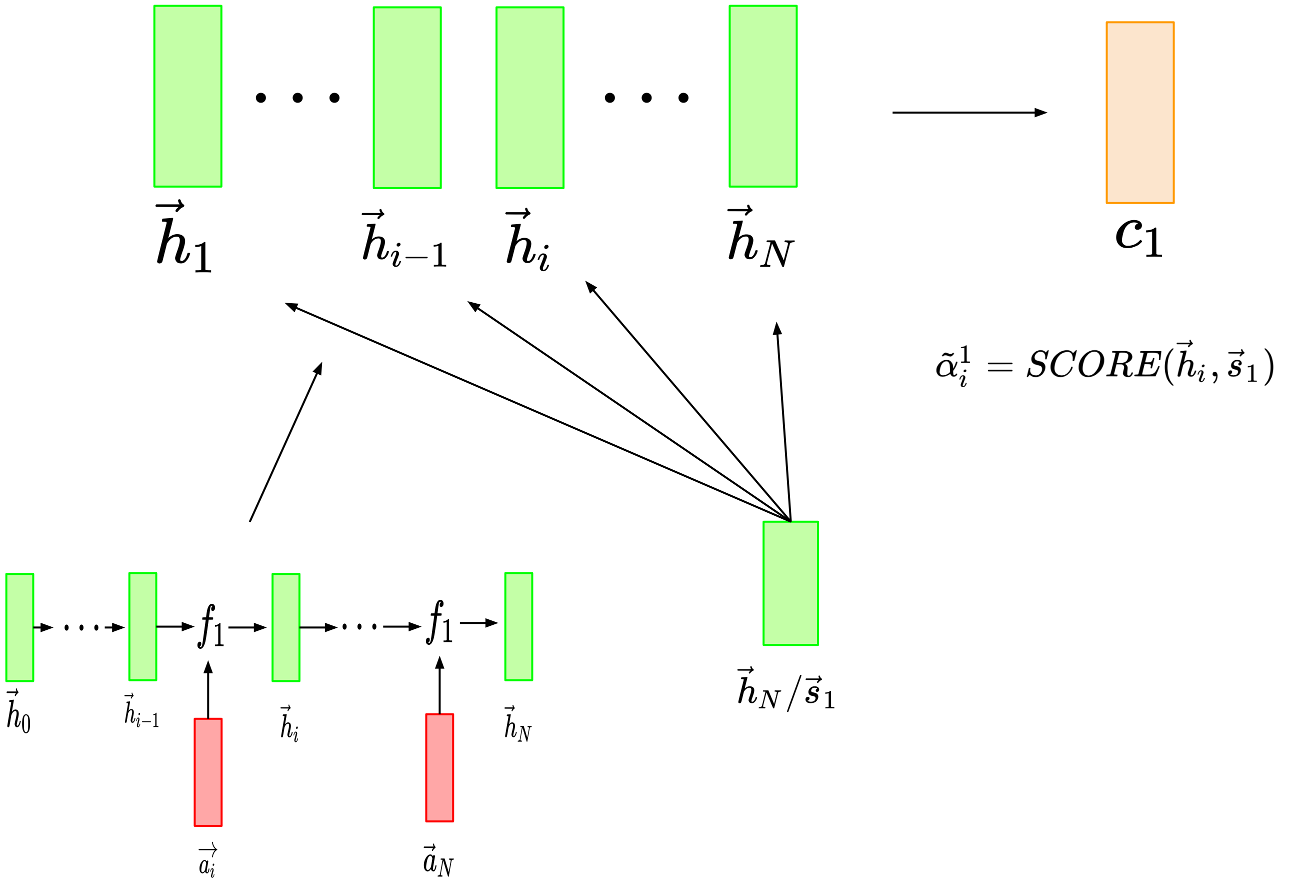
$$\sum_{i=1}^{N} \alpha_i^t = 1$$



Attention

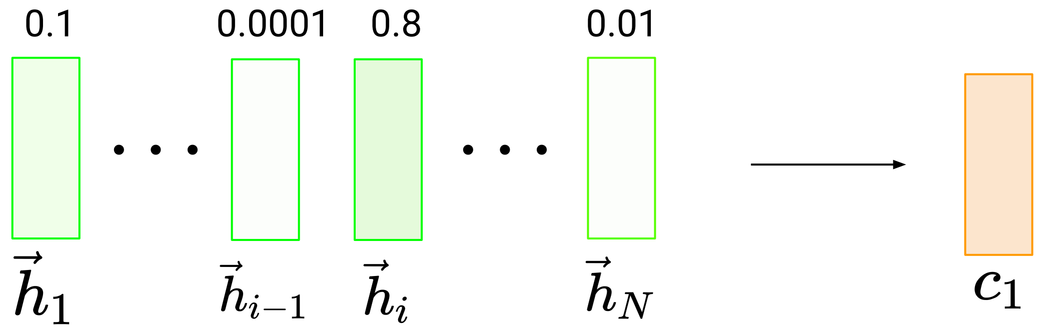
$\sum_{i=1}^{N} \alpha_i^t h_i = c_t$

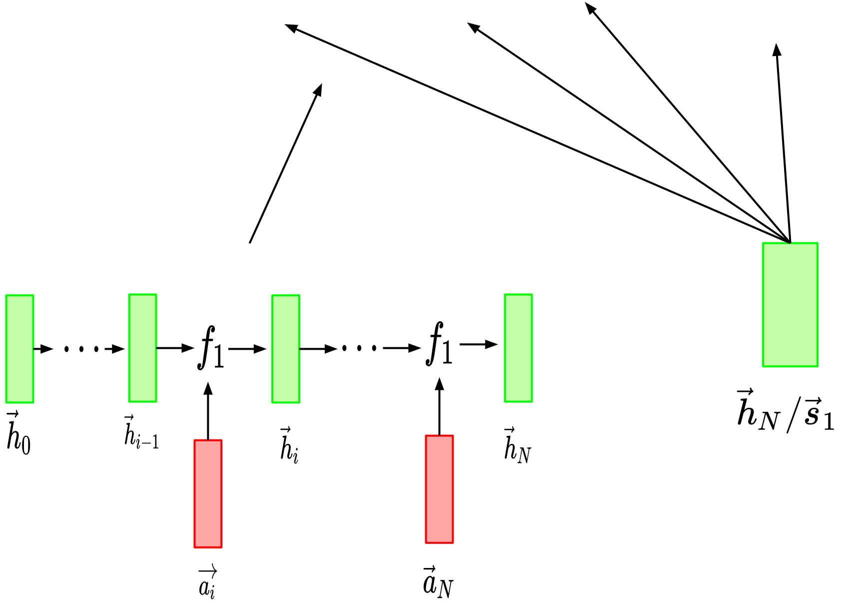
$\sum_{i=1}^{N} \alpha_i^t = 1$



Attention

Context Vector





$$\sum_{i=1}^{N} \alpha_i^t h_i = c_t$$

$$\sum_{i=1}^{N} \alpha_i^t = 1$$

Softmax

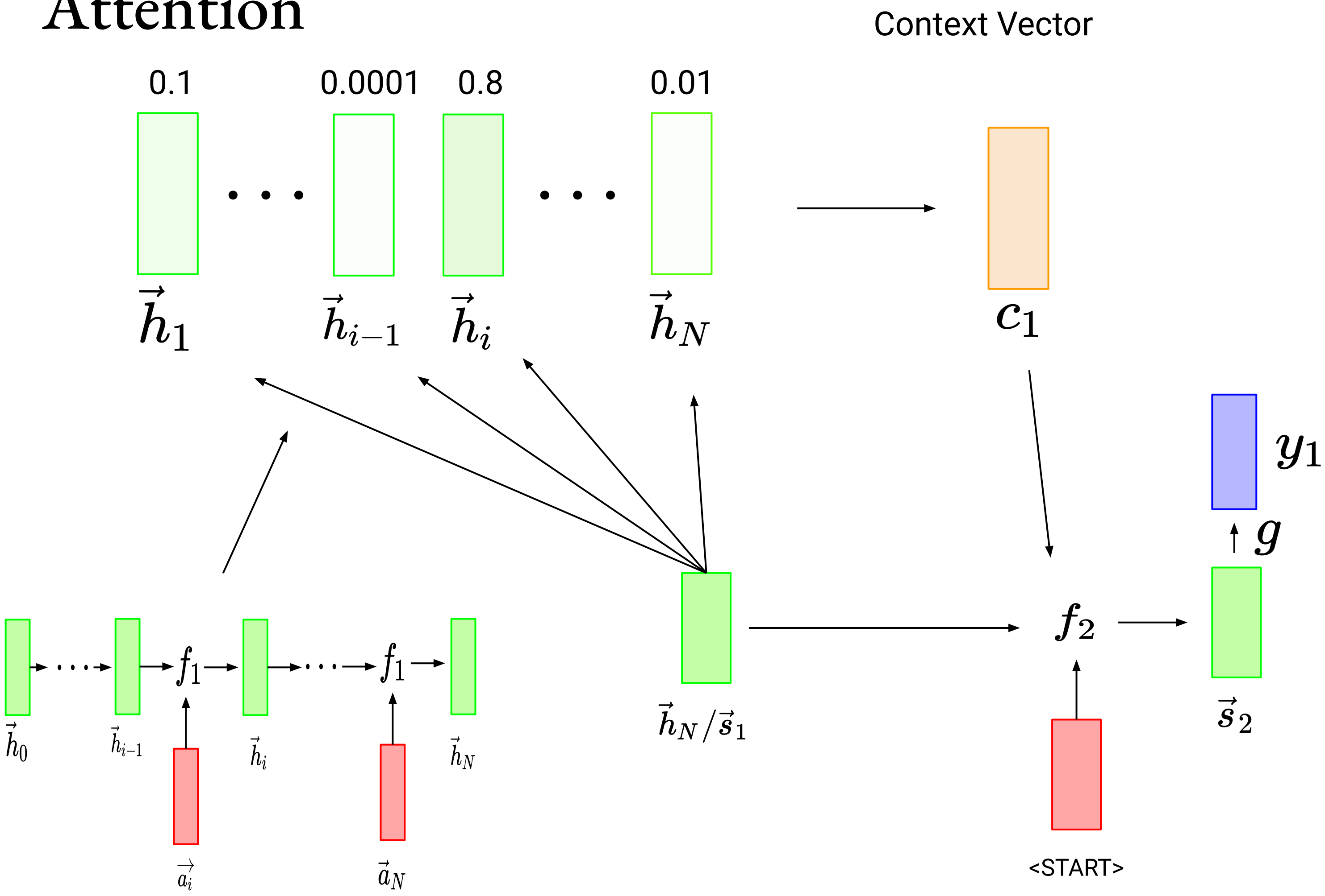
$$\tilde{\alpha}_i^1 = SCORE(\vec{h}_i, \vec{s}_1)$$

$$\alpha_i^1 = \frac{\exp(\tilde{\alpha}_i^1)}{\sum_{j=1}^{N} \exp(\tilde{\alpha}_j^1)}$$

Attention

$$\sum_{i=1}^{N} \alpha_i^t h_i = c_t$$

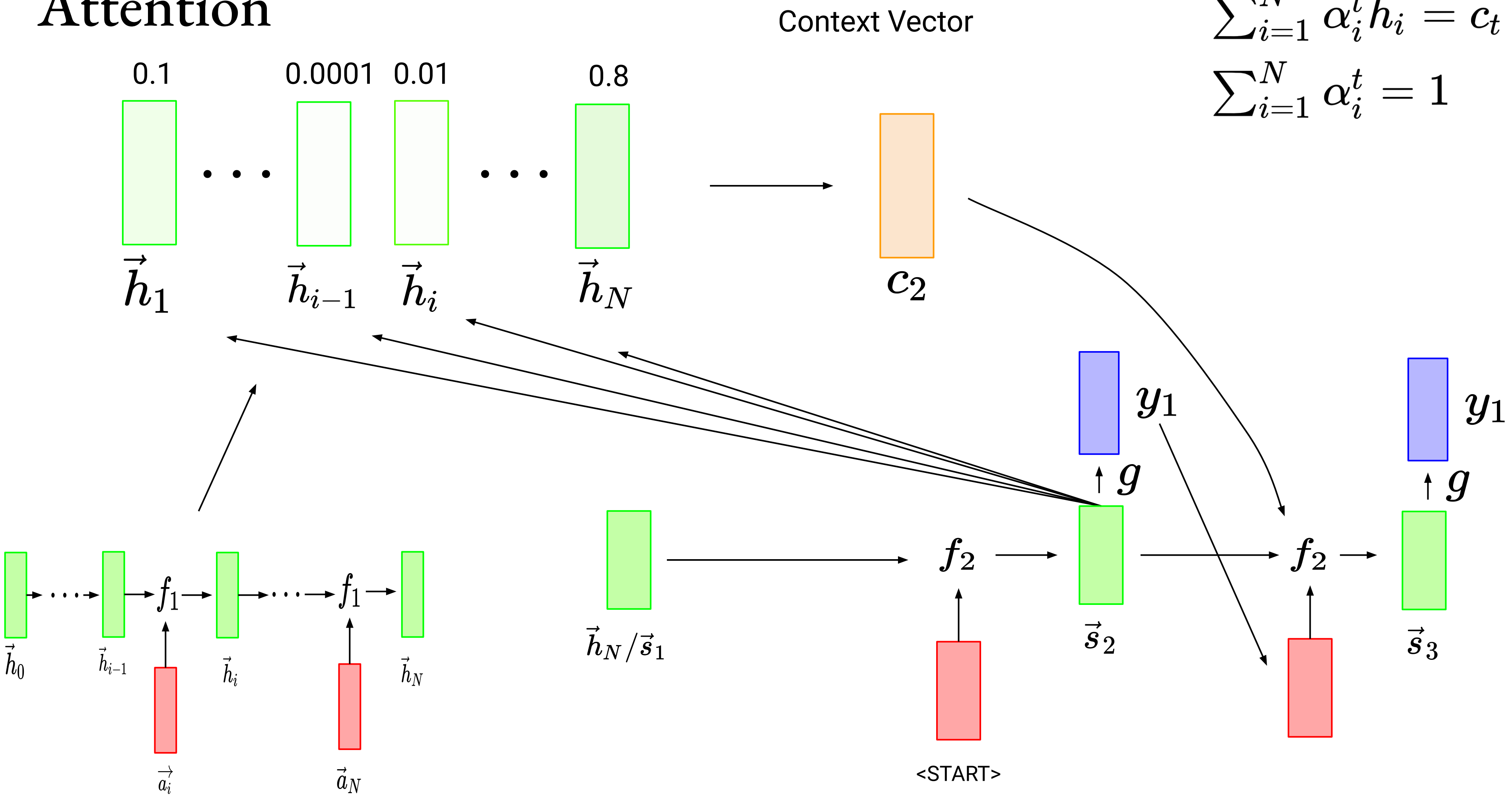
$$\sum_{i=1}^{N} \alpha_i^t = 1$$



Attention

$$\sum_{i=1}^{N} \alpha_i^t h_i = c_t$$

$$\sum_{i=1}^{N} \alpha_i^t = 1$$

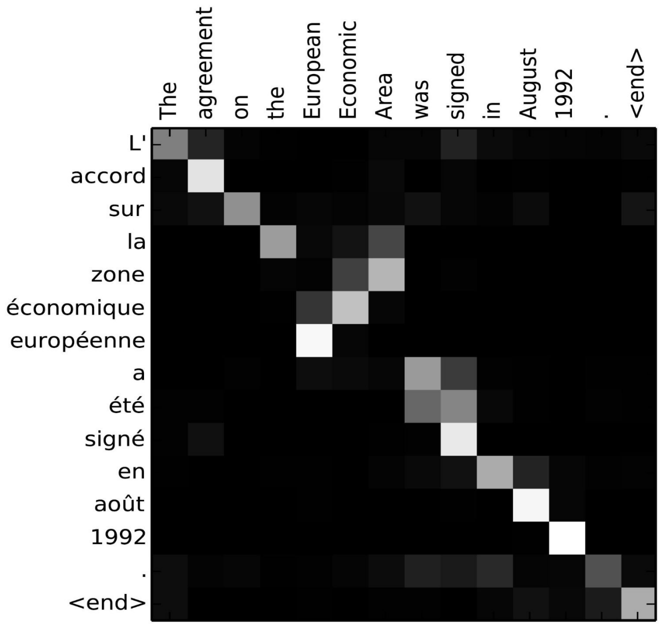


Attention

- The Attention Score function can be as simple as: $\vec{s}_t \cdot \vec{h}_i$
- Or a weight matrix can be introduced: $\vec{s}_t \cdot W_a \vec{h}_i$
    - (More can be found here)

Attention

- The Attention Score function can be as simple as: $\vec{s}_t \cdot \vec{h}_i$
- Or a weight matrix can be introduced: $\vec{s}_t \cdot W_a \vec{h}_i$
  - (More can be found [here](#))

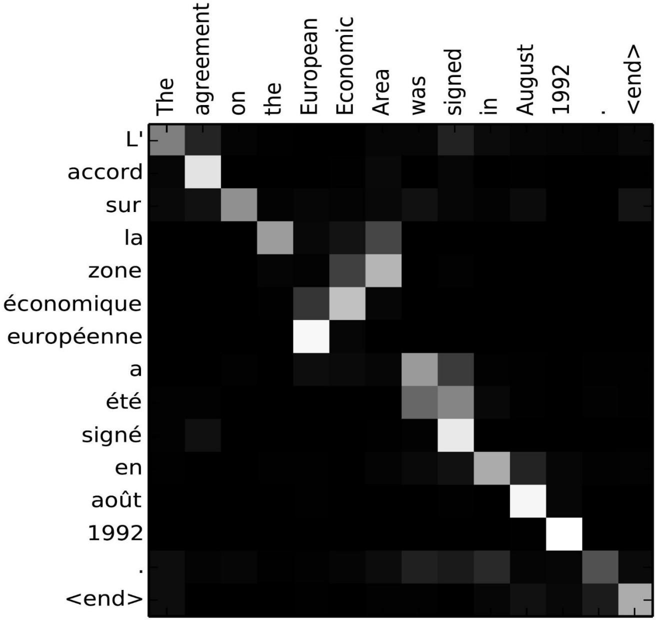


Bahdanau et al., 2015

- Attention Matrix
- Introduces some *Explainability*.

Attention

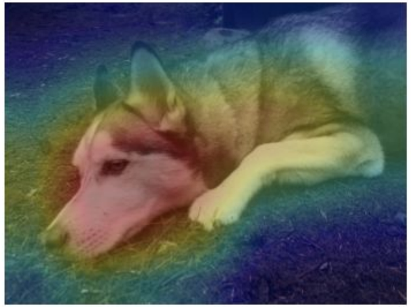
- The Attention Score function can be as simple as: $\vec{s}_t \cdot \vec{h}_i$
- Or a weight matrix can be introduced: $\vec{s}_t \cdot W_a \vec{h}_i$
    - (More can be found here)



Bahdanau et al., 2015

- Attention Matrix
- Introduces some *Explainability*.

Just because you pay attention to the right thing doesn't mean you are paying attention for the right *reason*.

	Test Image	Evidence for Animal Being a Siberian Husky	Evidence for Animal Being a Transverse Flute
Explanations Using Attention Maps			

Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead
By Cynthia Rudin 2018
[link]

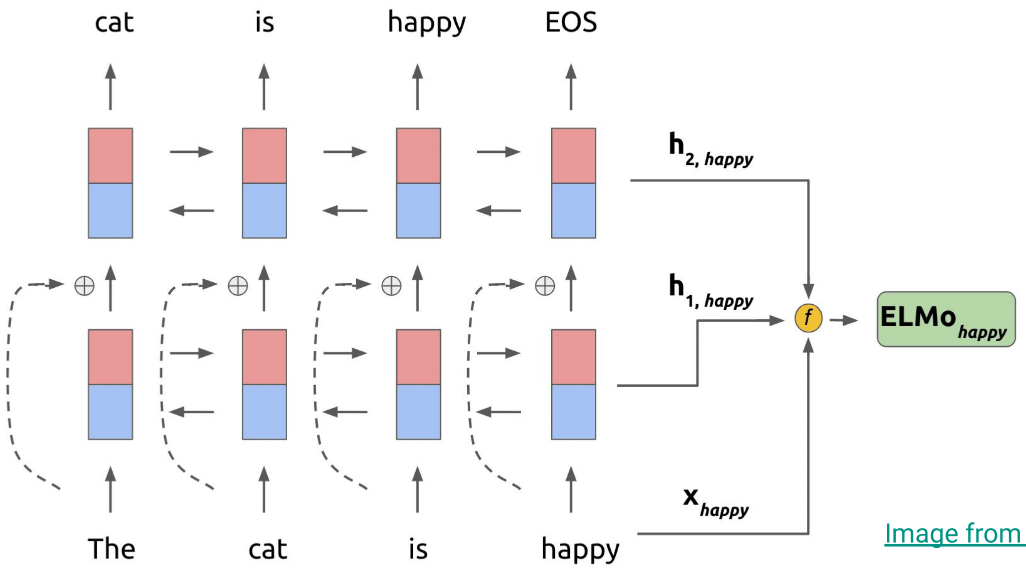
Just because you pay attention to the right thing doesn’t mean you are paying attention for the right *reason*.

Contextual Word Embeddings

- What about homonyms?
  - He took a *train*.
  - He likes to *train* in the mornings.

Contextual Word Embeddings

- ELMo: Embeddings from Language Models
  - Trained on self-supervised task such as predict the next word or Part of Speech tagging
  - Creates contextual word embeddings



Idea: Stack bi-directional LSTMs to produce multiple embeddings of words that use the context.

- (Can get more complicated, but that's the core)

Image from here

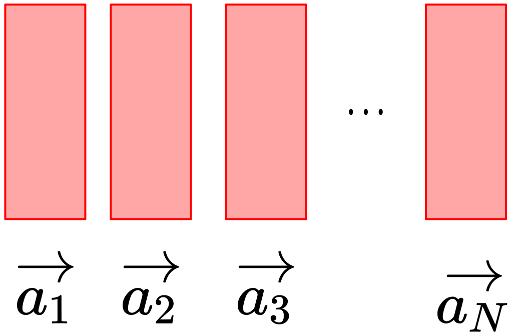
Self-Attention

- Goal: Only use Attention mechanism to encode a sequence of tokens.
  - Generate contextual embeddings of the tokens without RNNs!

Self-Attention

$$W_Q, W_K, W_V$$

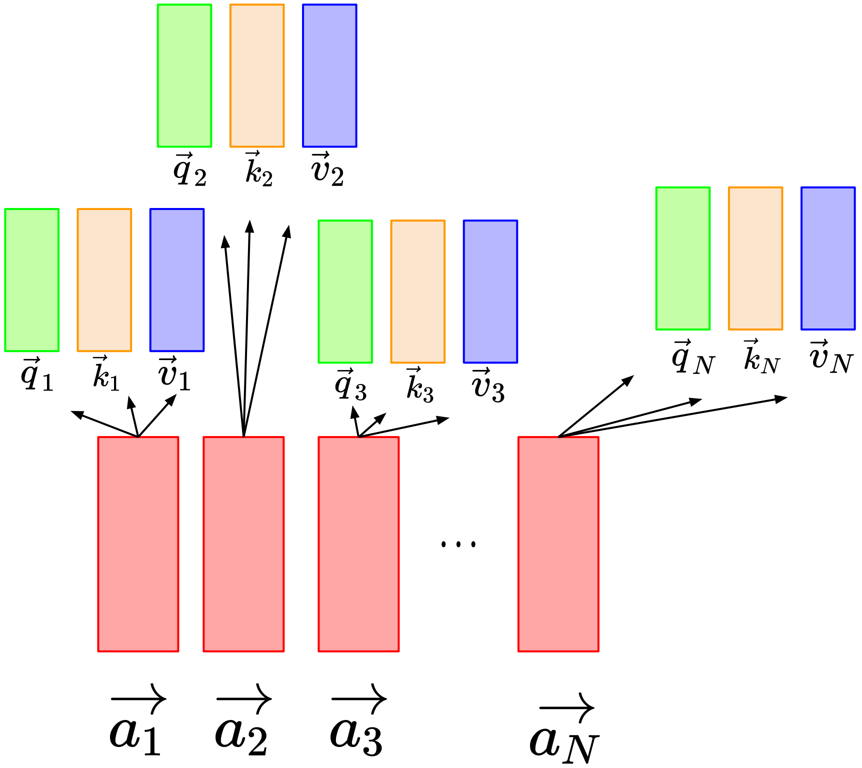
- Initial Embeddings
- 3 weight matrices: Query, Key Value



Self-Attention

W_Q, W_K, W_V

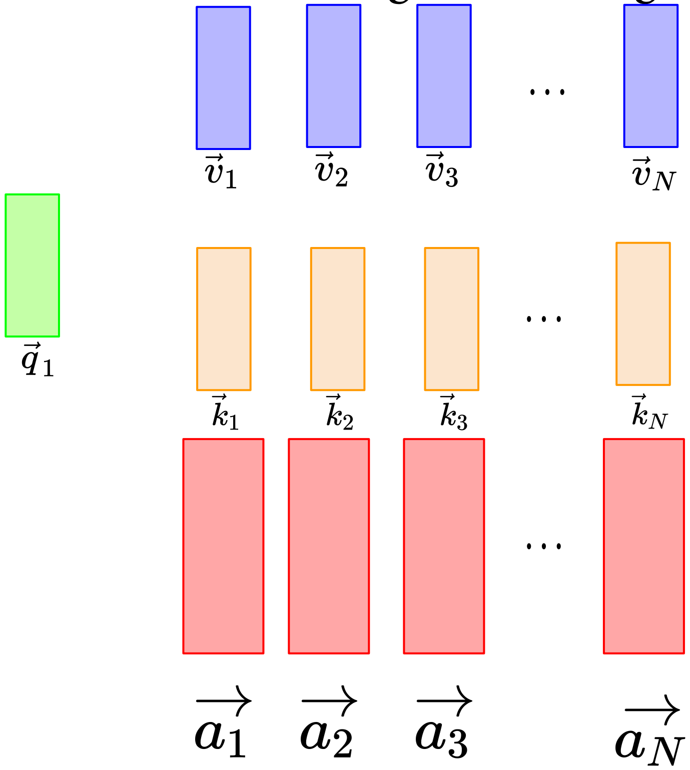
- Each token gets a Query, Key, and Value vector.



Self-Attention

W_Q, W_K, W_V

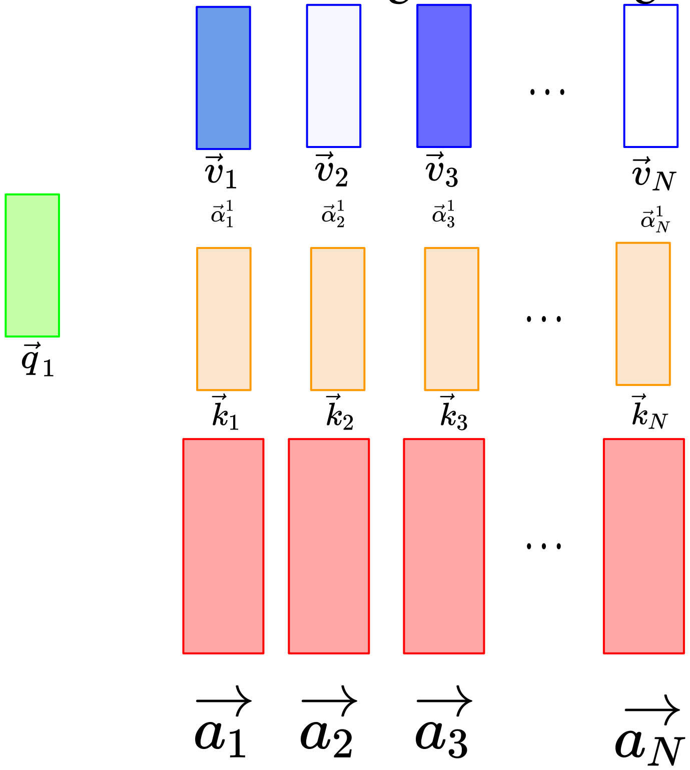
- For each vector we use *its query* to explore *all the keys* which generate weights.
- The resulting embedding is the weighted sum of all the value vectors.



Self-Attention

W_Q, W_K, W_V

- For each vector we use *its query* to explore *all the keys* which generate weights.
- The resulting embedding is the weighted sum of all the value vectors.

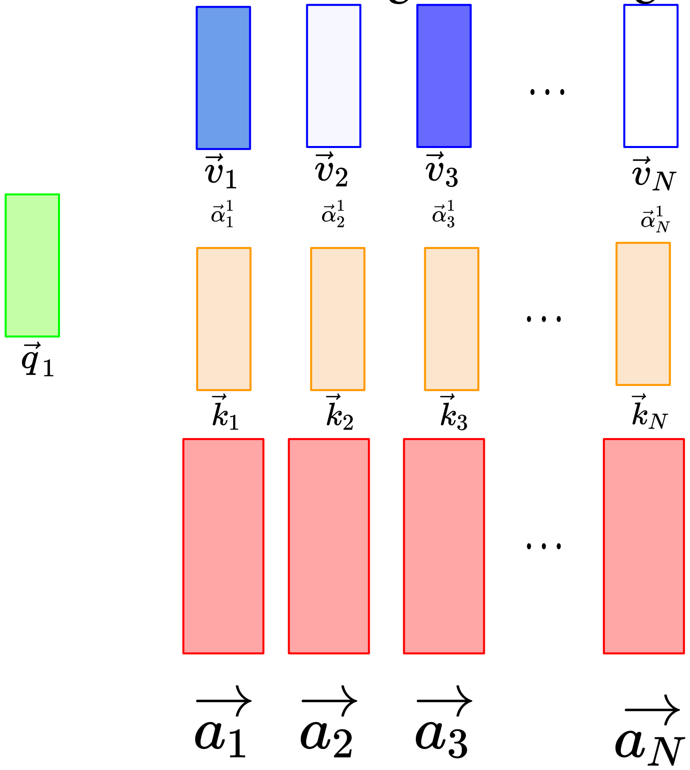


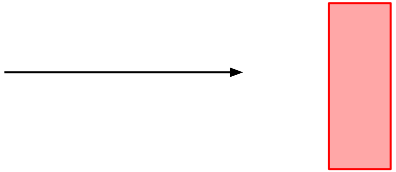
- We first take the dot product of q1 with every key vector.
- We get the weights by using a softmax layer.
- Sometimes there is a constant multiplier in there as well (for stability purposes)

Self-Attention

W_Q, W_K, W_V

- For each vector we use *its query* to explore *all the keys* which generate weights.
- The resulting embedding is the weighted sum of all the value vectors.





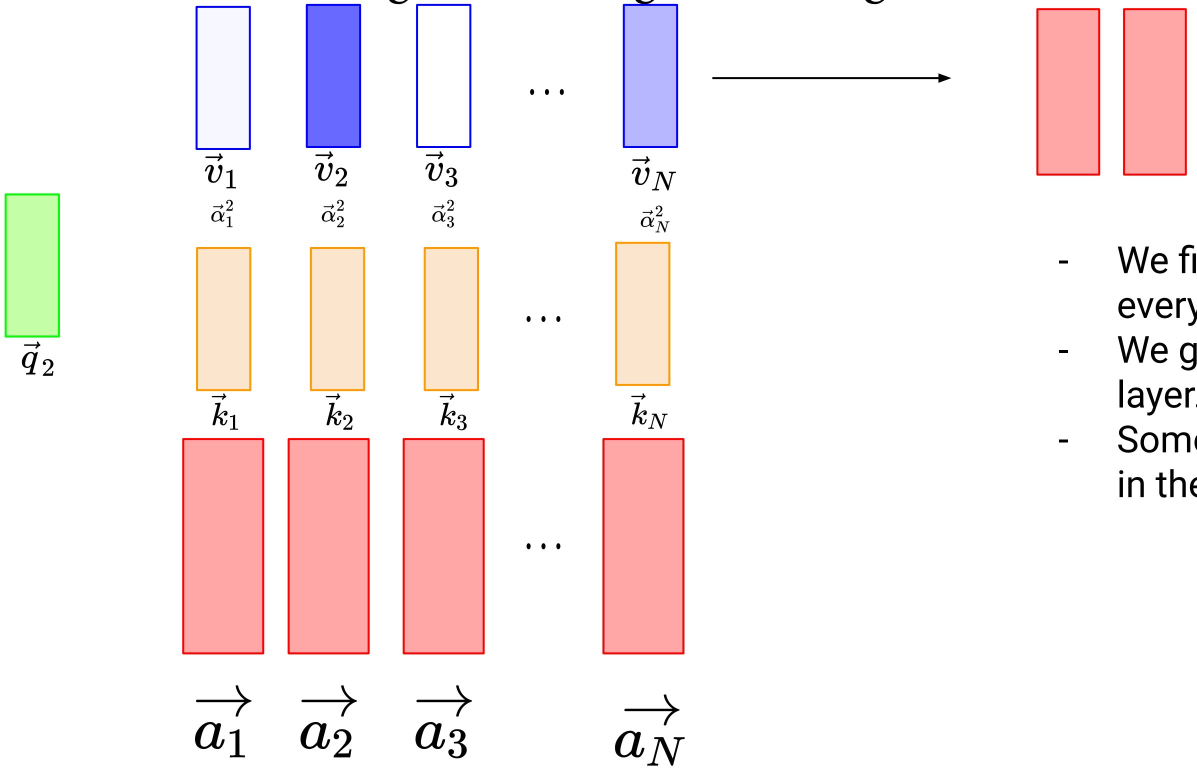
- We first take the dot product of q1 with every key vector.
- We get the weights by using a softmax layer.
- Sometimes there is a constant multiplier in there as well (for stability purposes)

$$\sum_{i=1}^{N} \alpha_i^t \vec{v}_i$$

Self-Attention

W_Q, W_K, W_V

- For each vector we use *its query* to explore *all the keys* which generate weights.
- The resulting embedding is the weighted sum of all the value vectors.

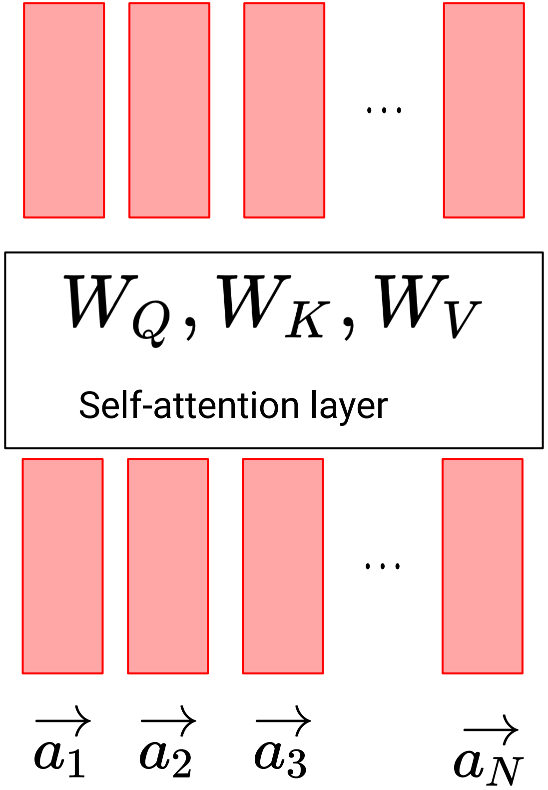


- We first take the dot product of q1 with every key vector.
- We get the weights by using a softmax layer.
- Sometimes there is a constant multiplier in there as well (for stability purposes)

$$\sum_{i=1}^{N} \alpha_i^t \vec{v}_i$$

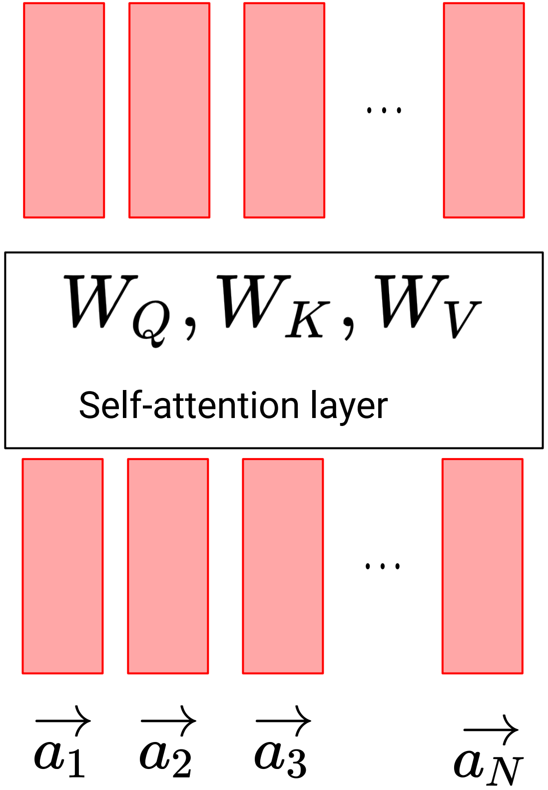
Self-Attention

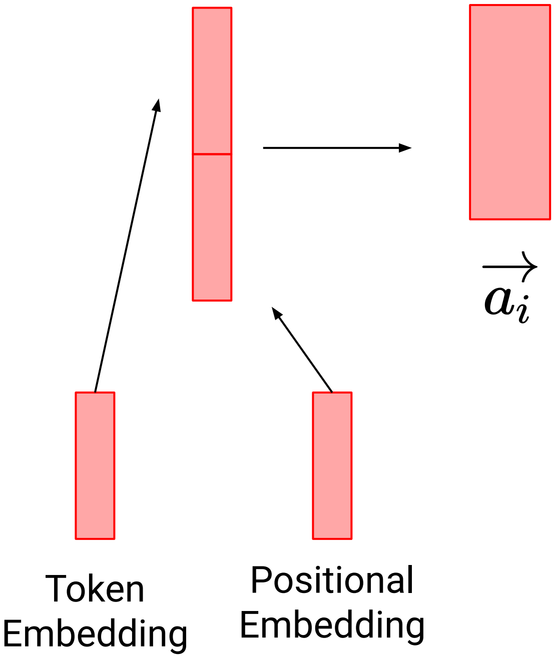
- Contextual Embeddings!



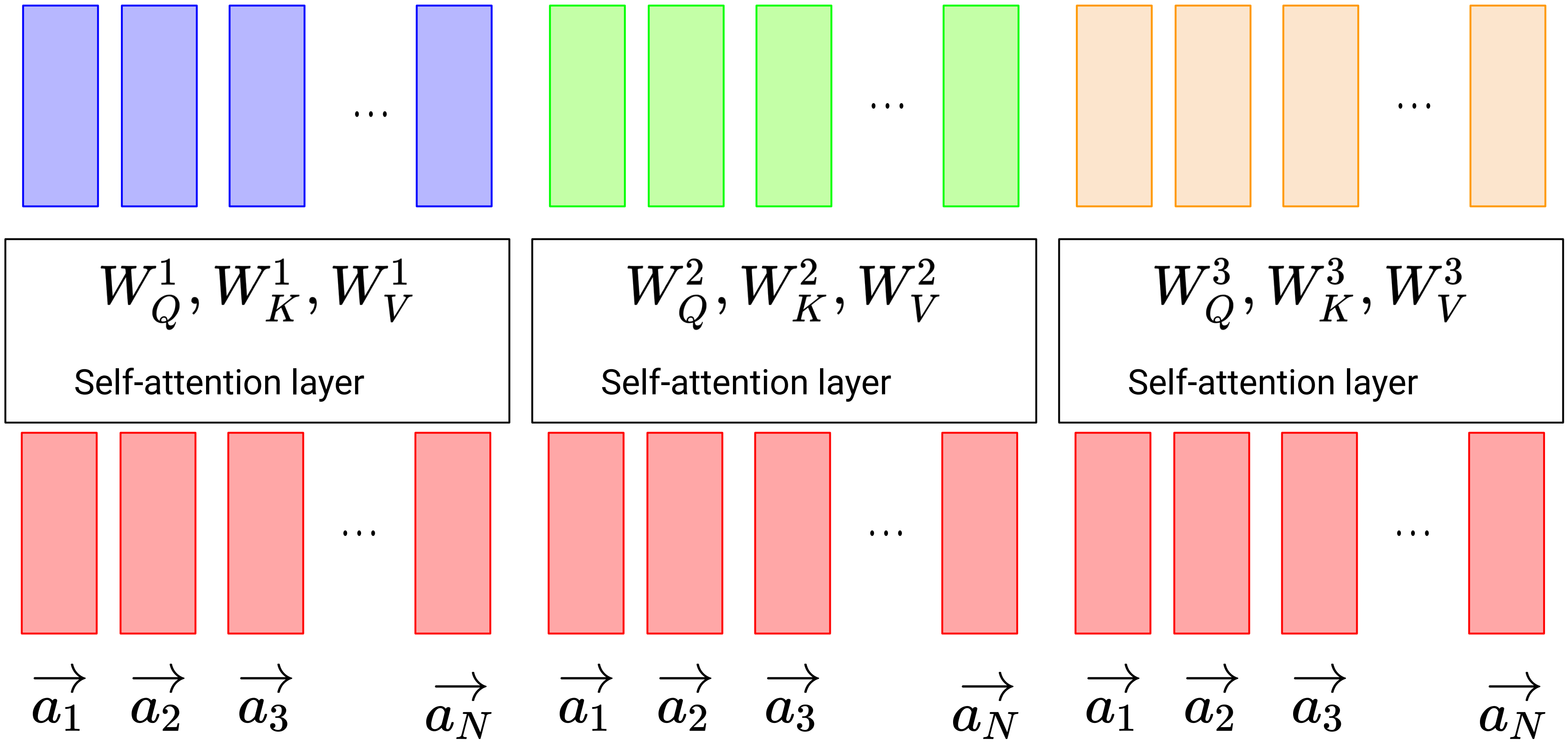
Self-Attention

- Sequence Agnostic! Often Positional Embeddings are added here as well.

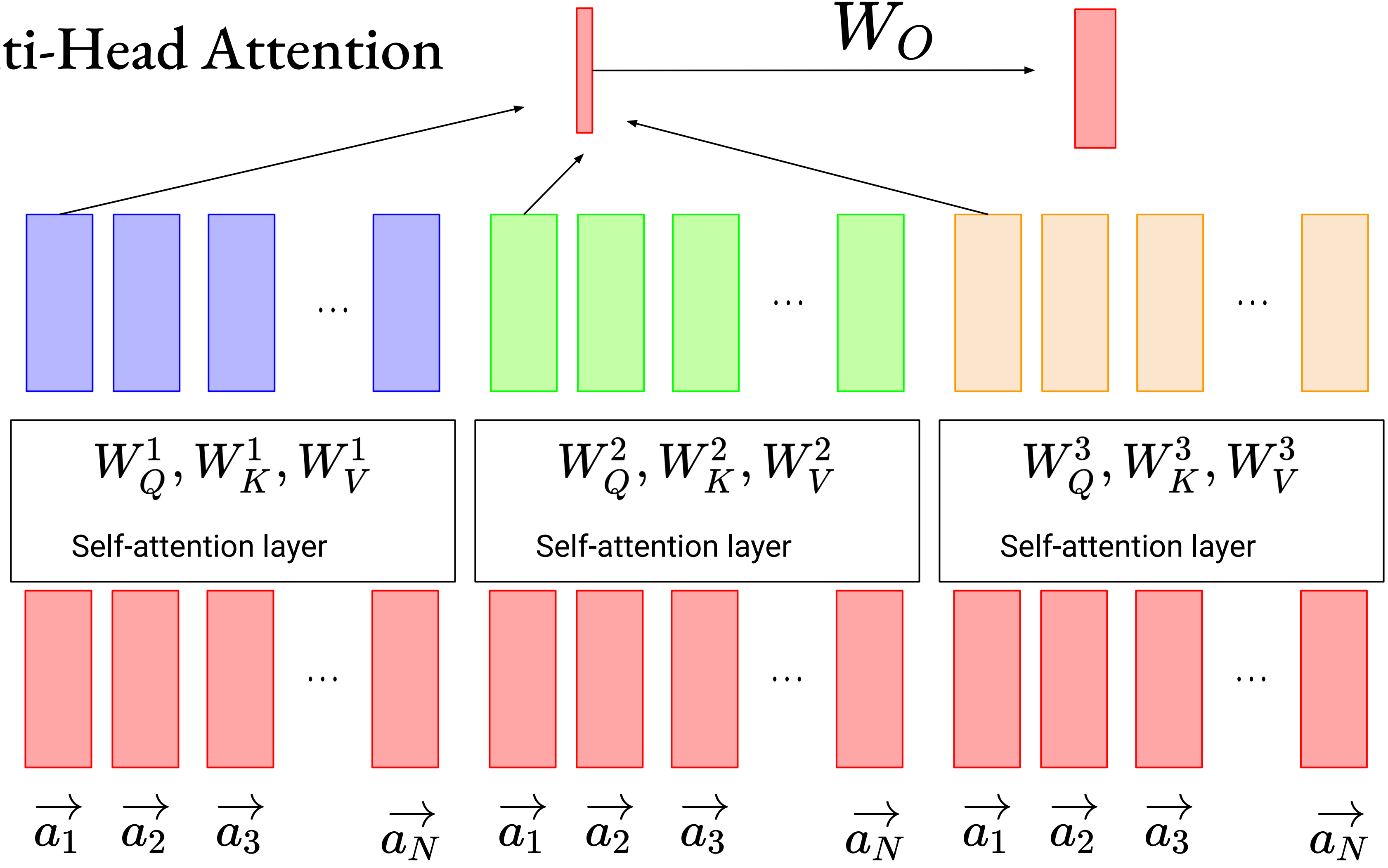




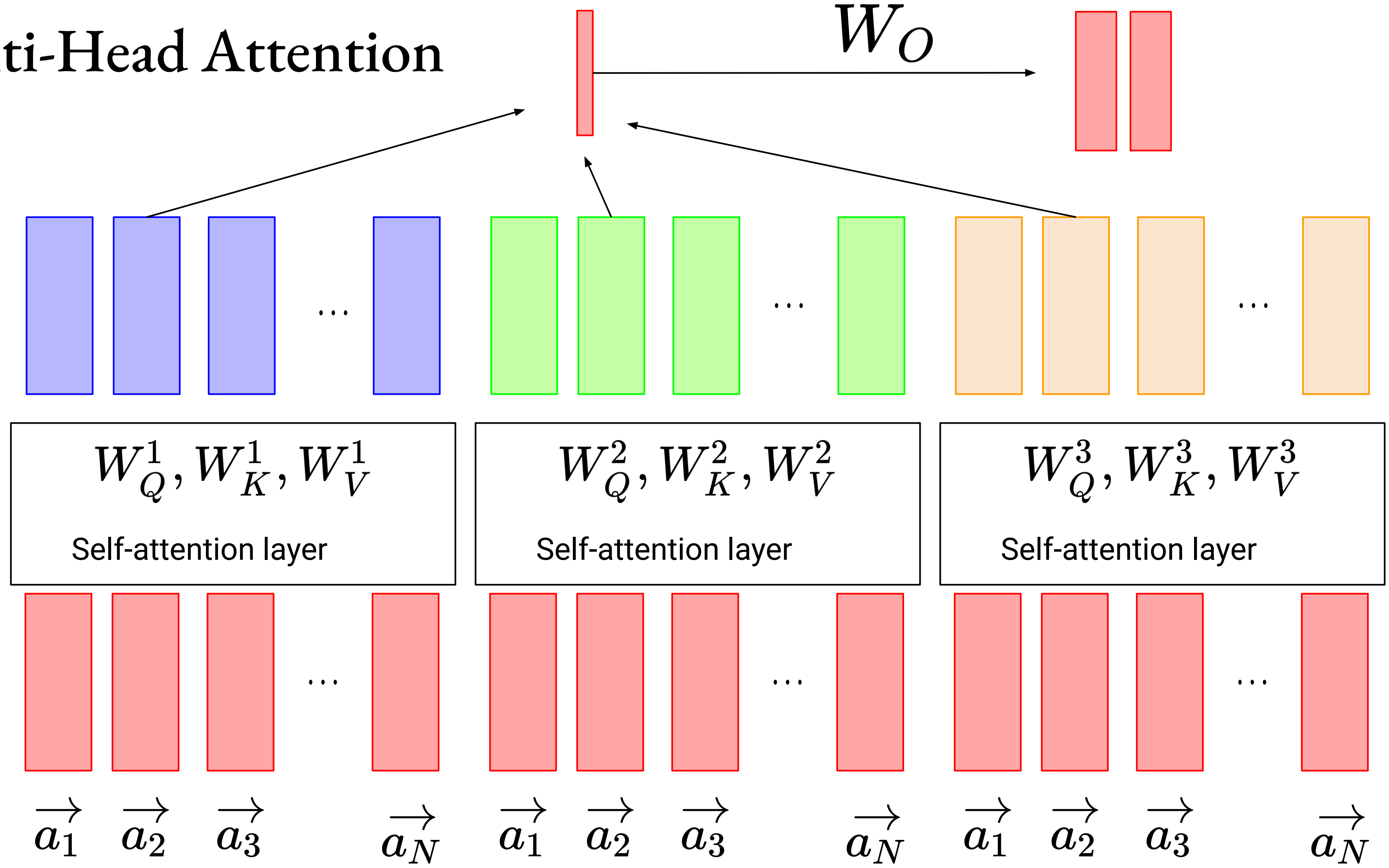
Multi-Head Attention



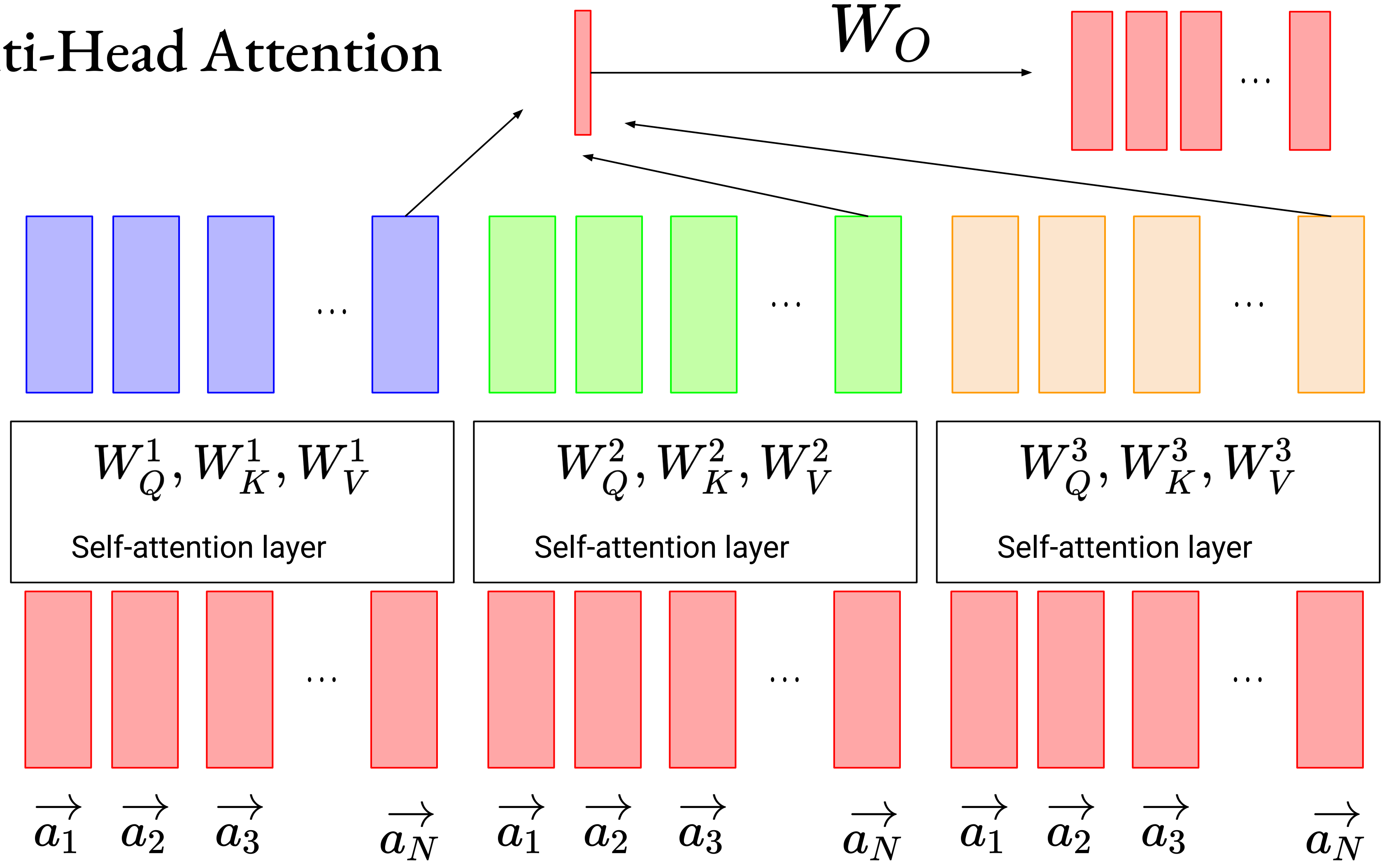
Multi-Head Attention



Multi-Head Attention

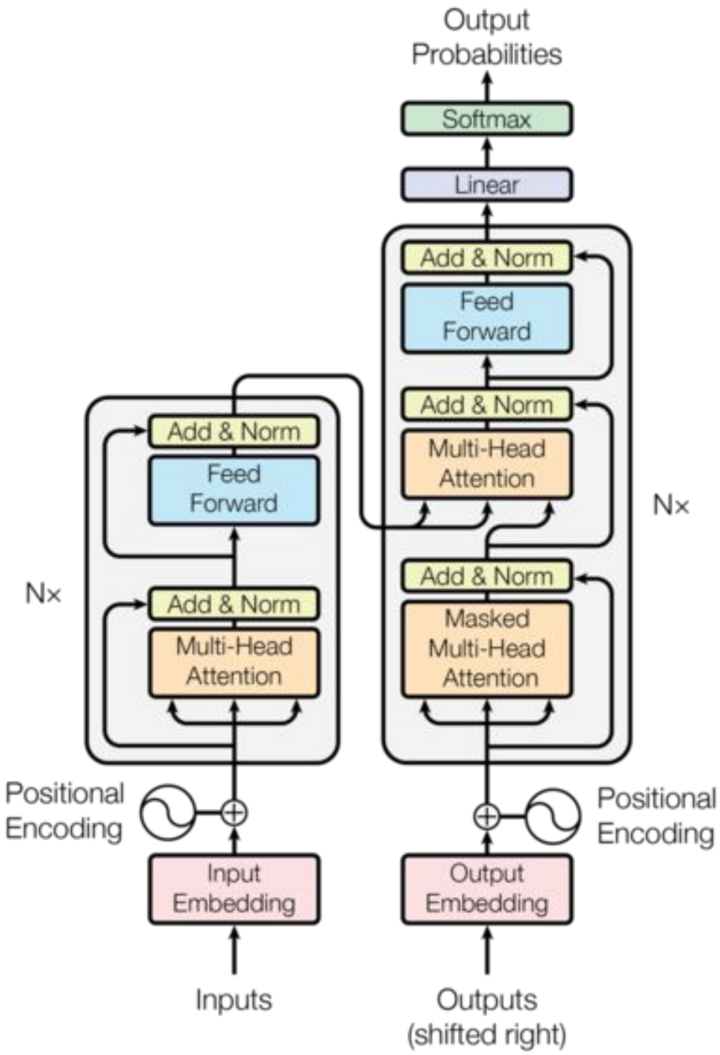


Multi-Head Attention



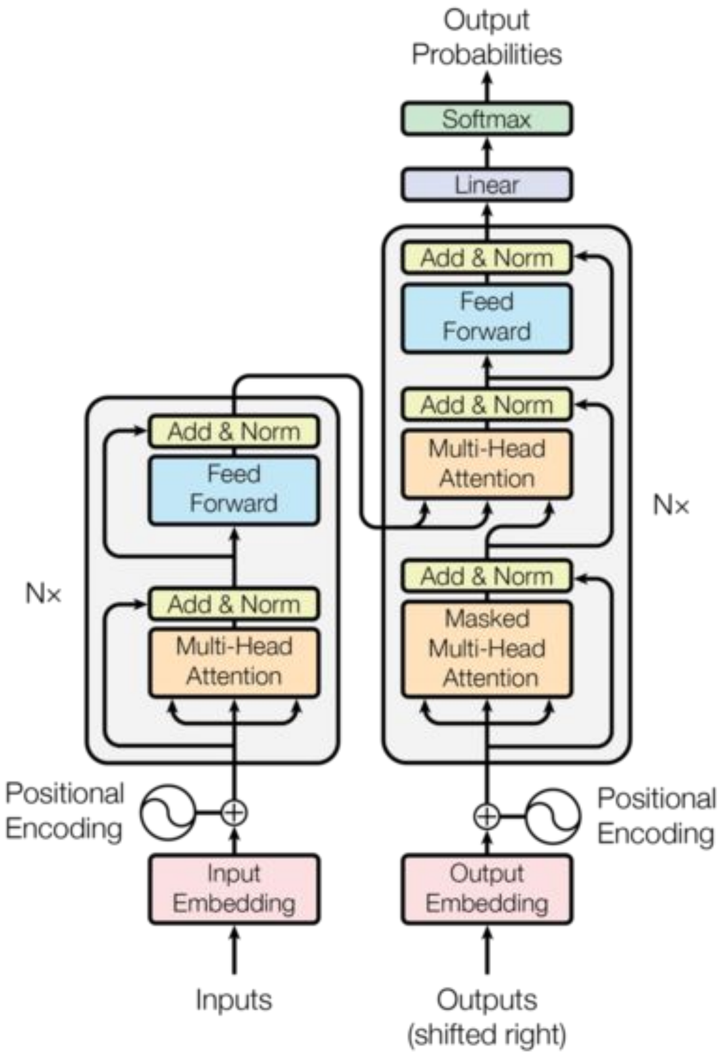
Transformer

- You can now understand everything in this famous diagram from the "Attention is All You Need" paper.



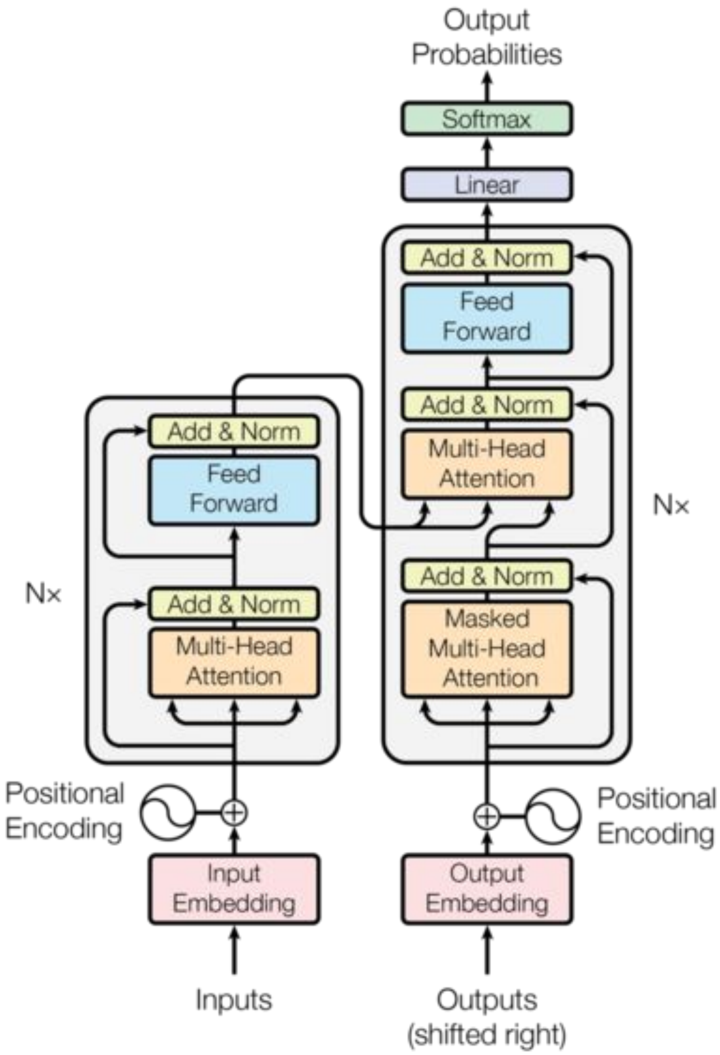
Transformer

- You can now understand everything in this famous diagram from the "Attention is All You Need" paper.
  - "Add & Norm" is like a residual connection
  1. Add the initial sequence embedding to the output of the MHA layer.
  2. Pass it through a Layer Normalization.



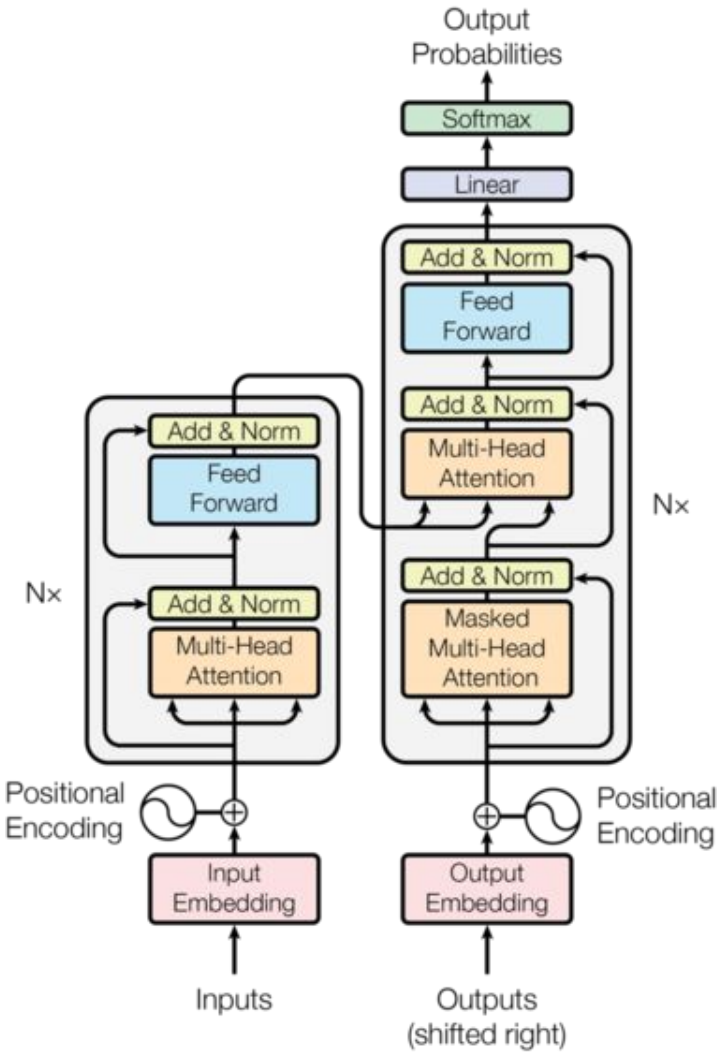
Transformer

- You can now understand everything in this famous diagram from the "Attention is All You Need" paper.
  - "Add & Norm" is like a residual connection
  - That output gets put into a feed forward layer (and then another residual connection)

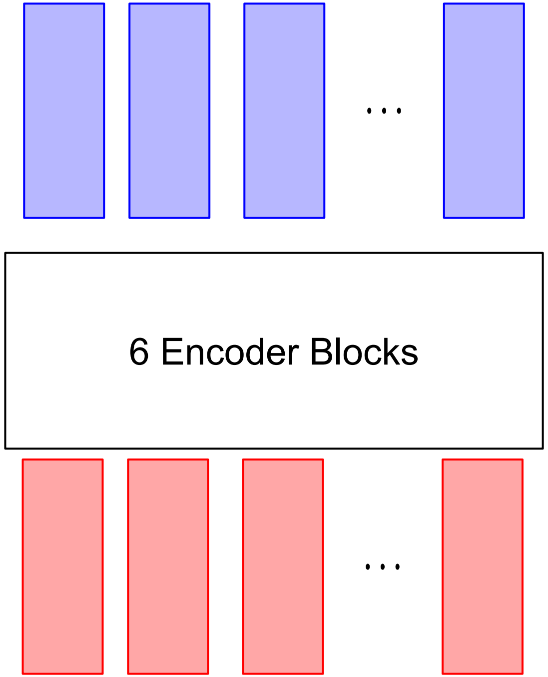


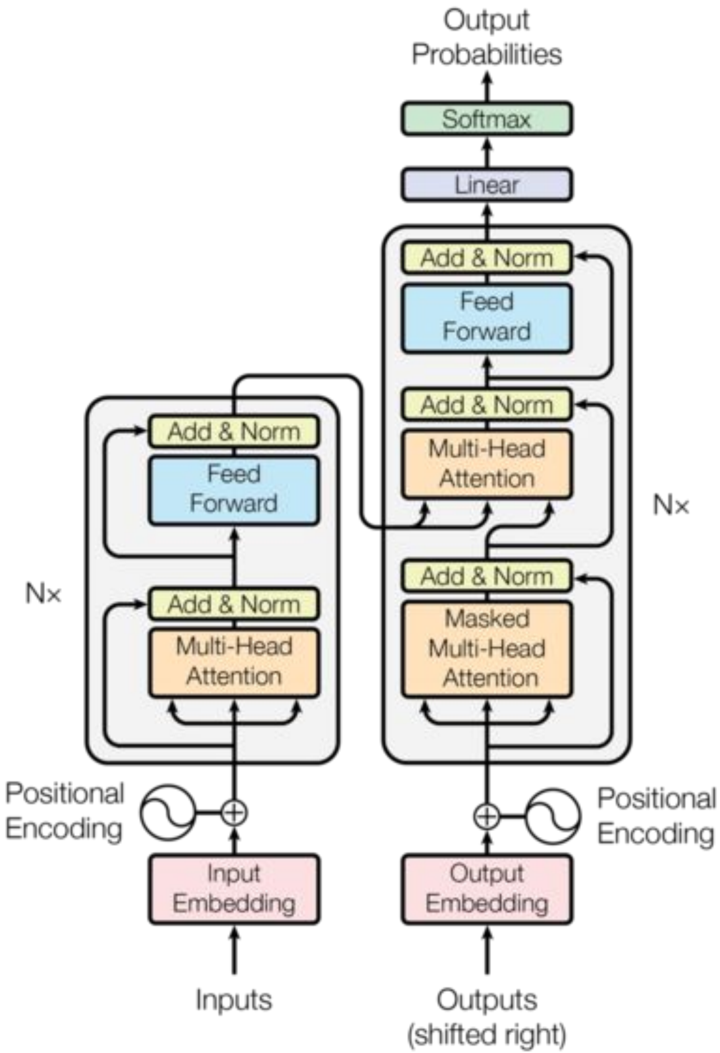
Transformer

- You can now understand everything in this famous diagram from the “Attention is All You Need” paper.
  - “Add & Norm” is like a residual connection
  - That output gets put into a feed forward layer (and then another residual connection)
  - Original model has 6 Transformer encoder layers stacked on one another.

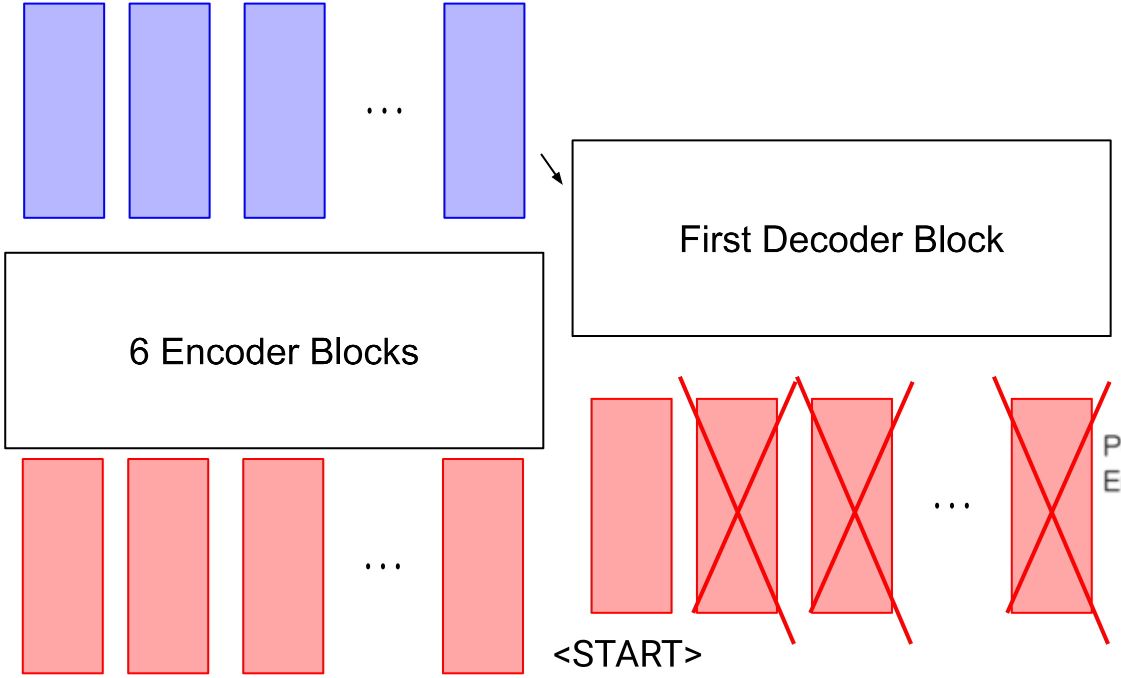


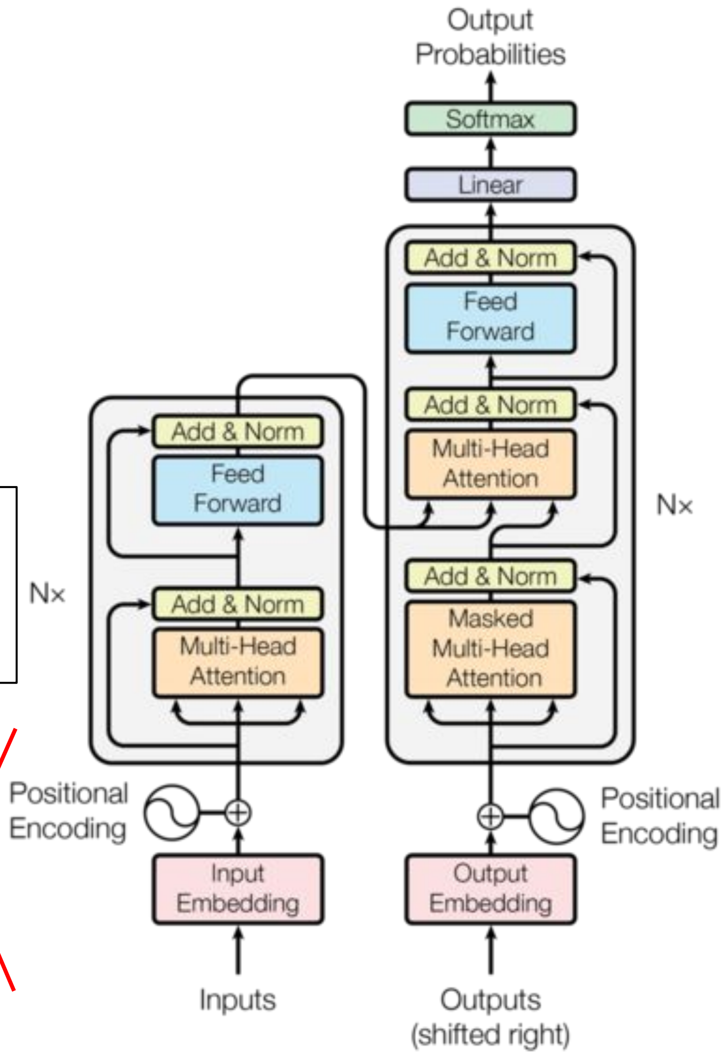
Transformer





Transformer

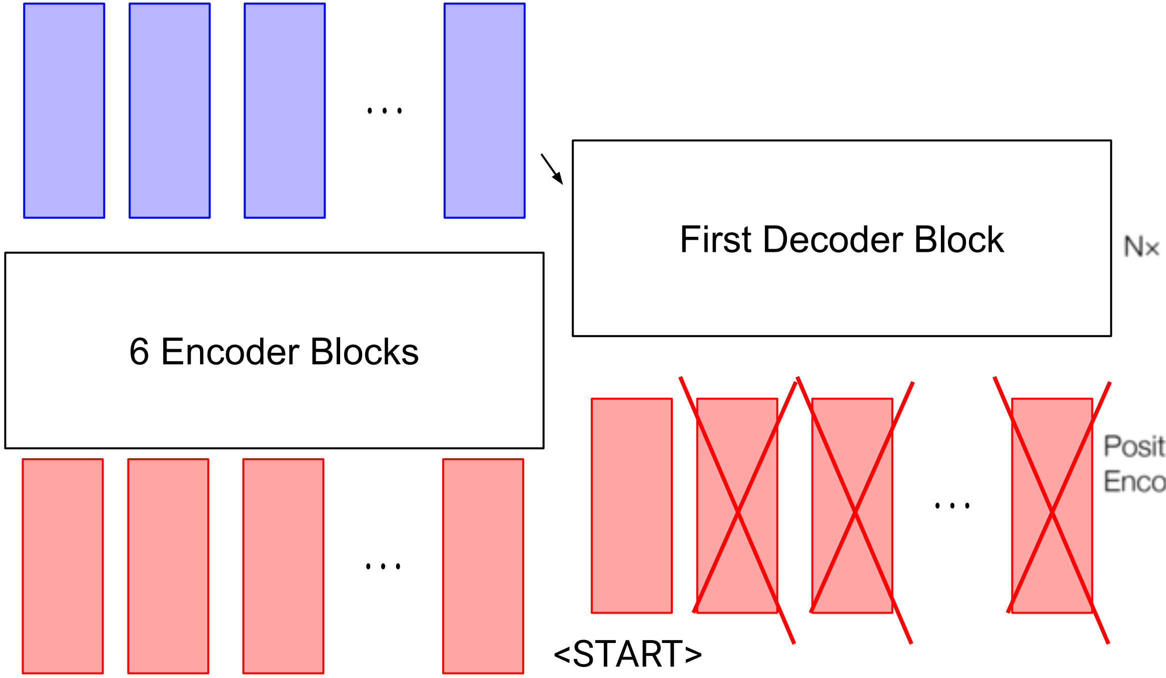


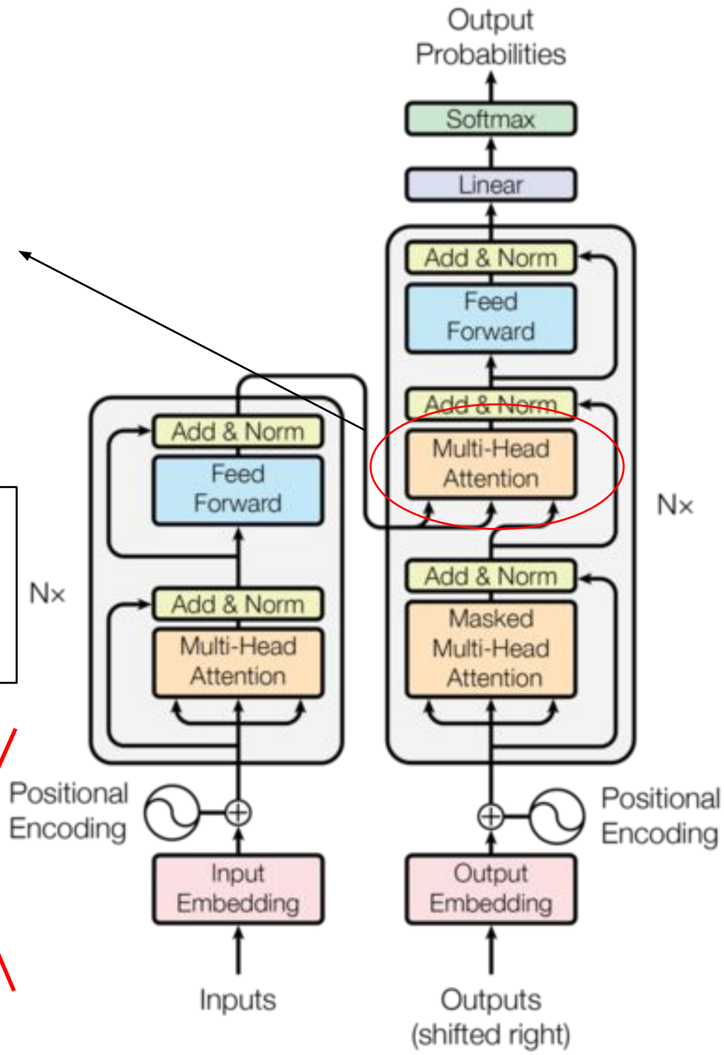


Transformer

Keys and values come from encoder stack output

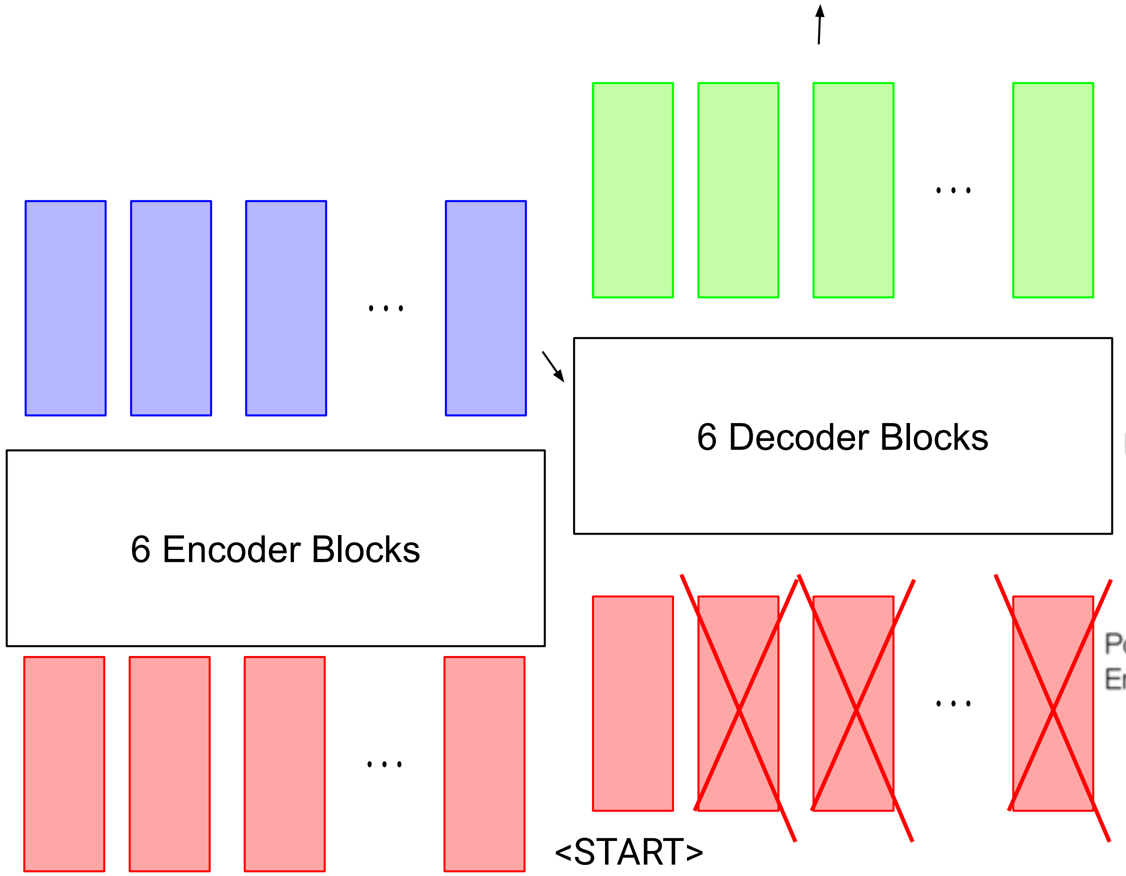
Queries come from decoder

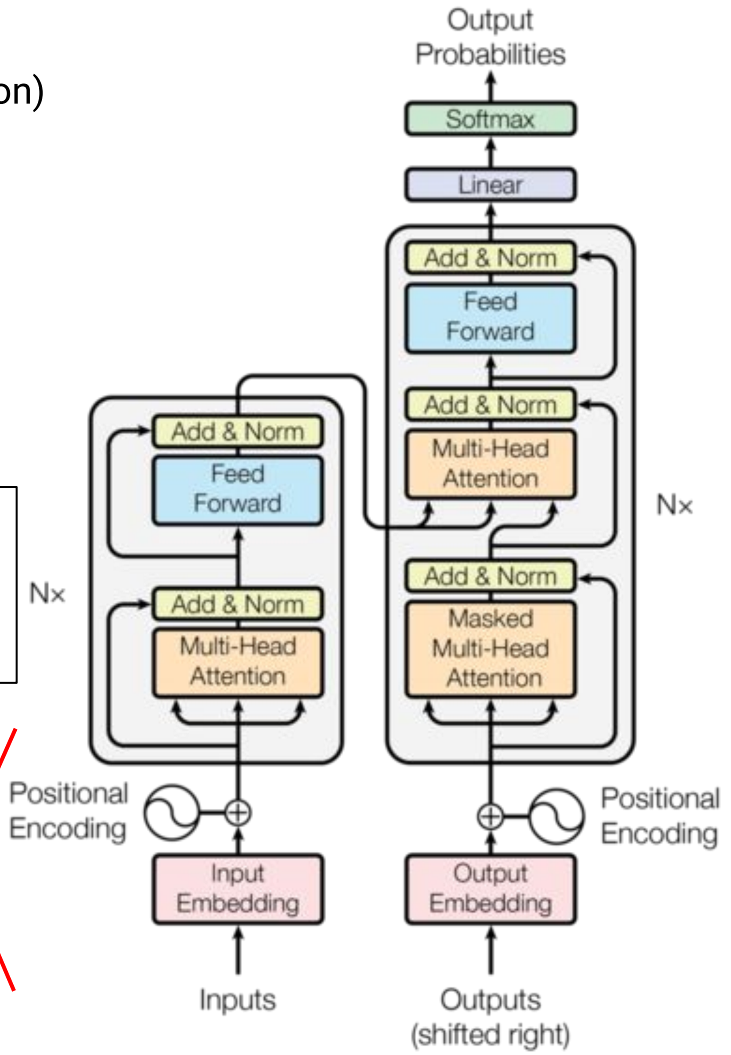




Transformer

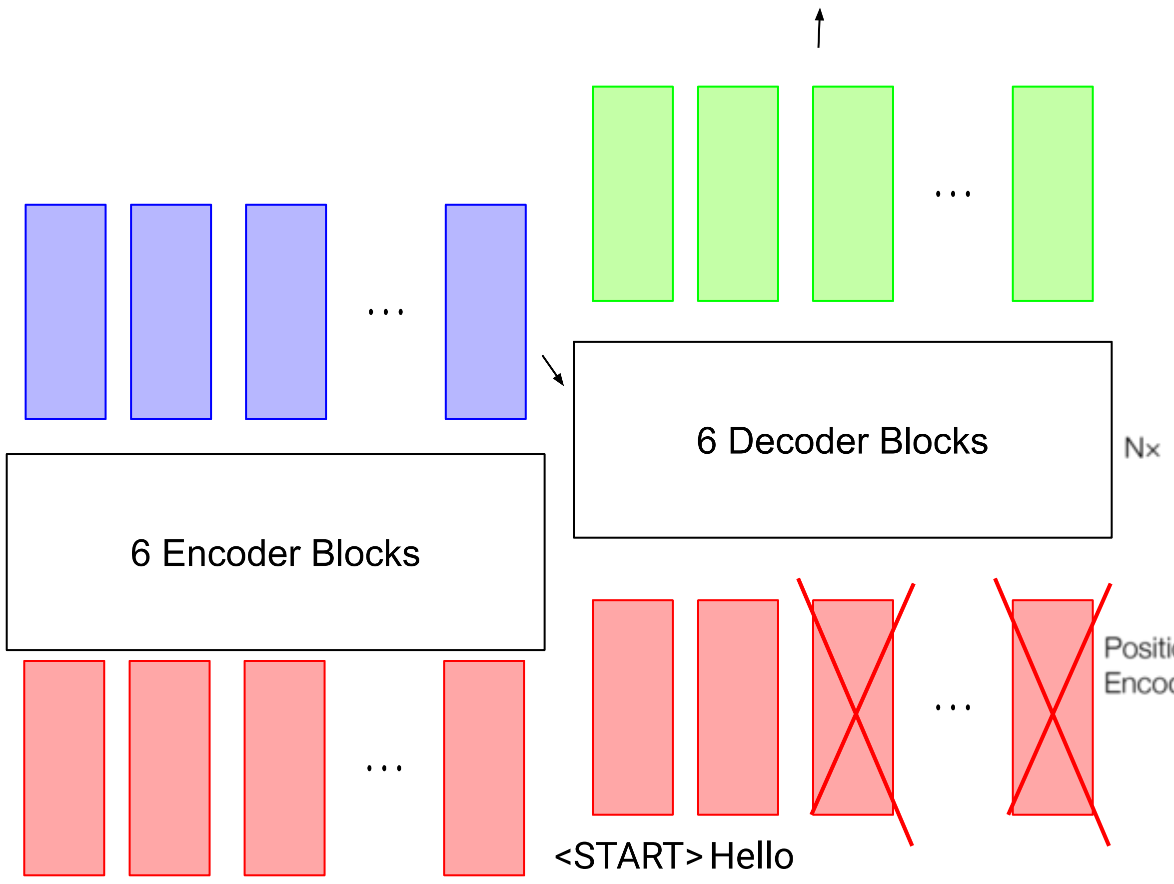
Dictionary-sized vector of probabilities (next word prediction)

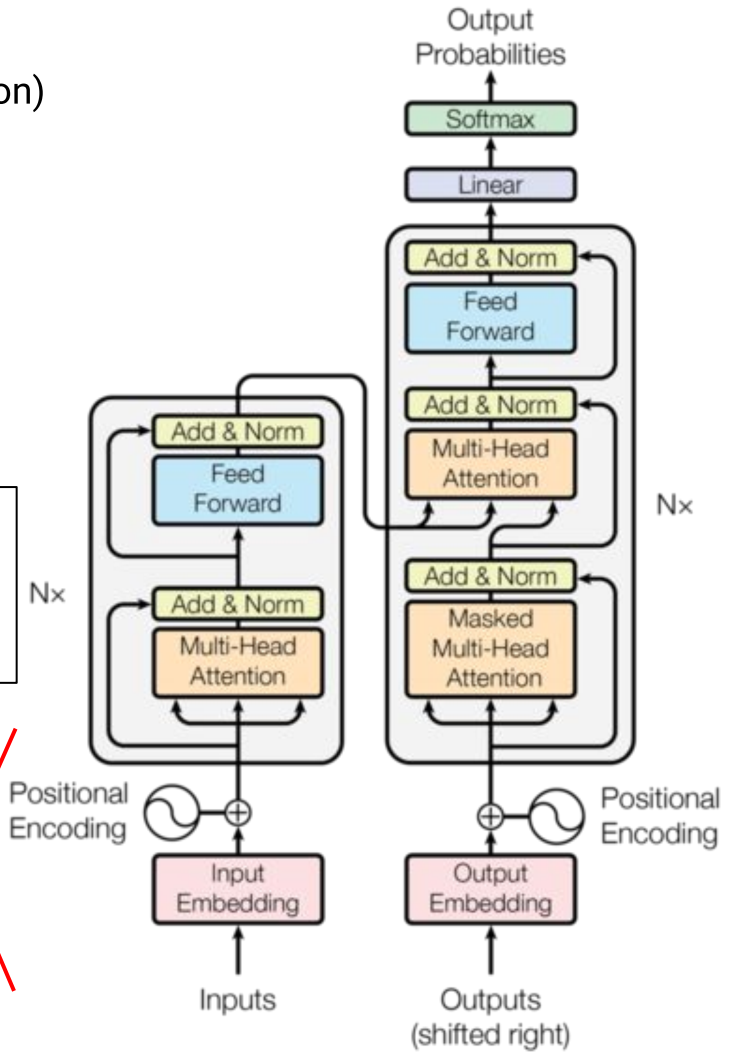




Transformer

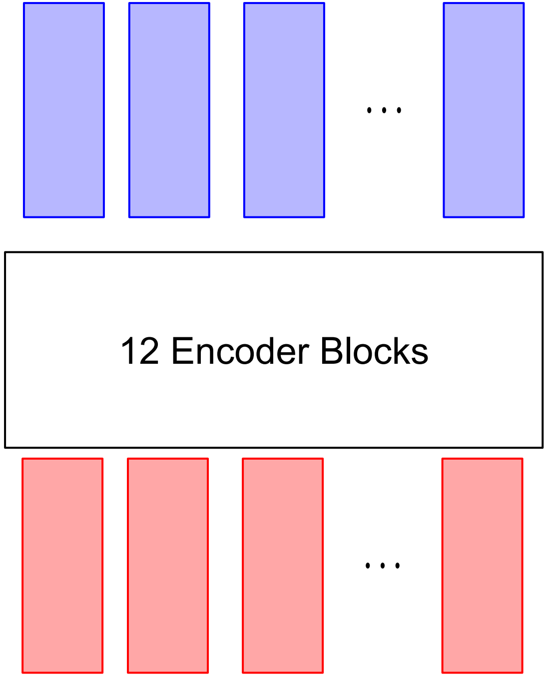
Dictionary-sized vector of
probabilities (next word prediction)





Self-Supervised Learning

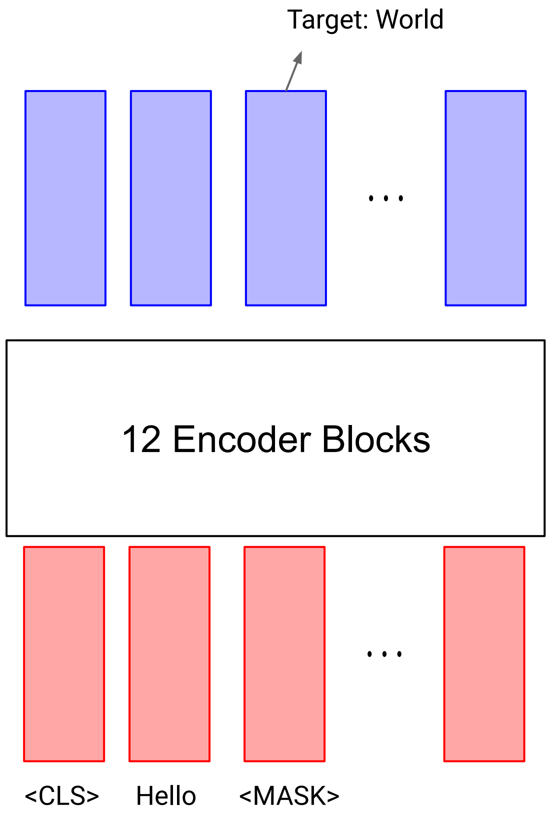
- BERT: Bidirectional Encoder Representations from Transformers
  - Encoder-only model
  - 12 Encoders



Self-Supervised Learning

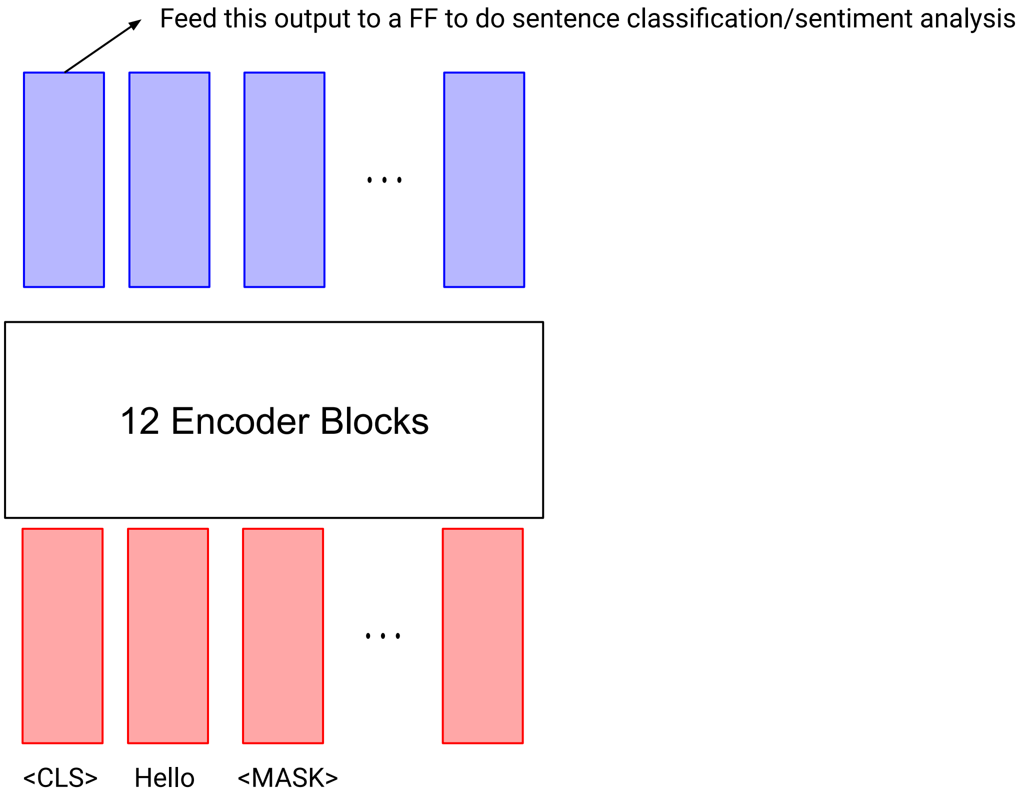
- BERT: Bidirectional Encoder Representations from Transformers
  - Encoder-only model
  - 12 Encoders
- Self-supervised Task #1

1. Take a mountain of text.
2. Mask ~15% of the words
3. Try to predict the masked words.



Self-Supervised Learning

- BERT: Bidirectional Encoder Representations from Transformers
  - Encoder-only model
  - 12 Encoders
- Self-supervised Task #1
1. Take a mountain of text.
2. Mask ~15% of the words
3. Try to predict the masked words.



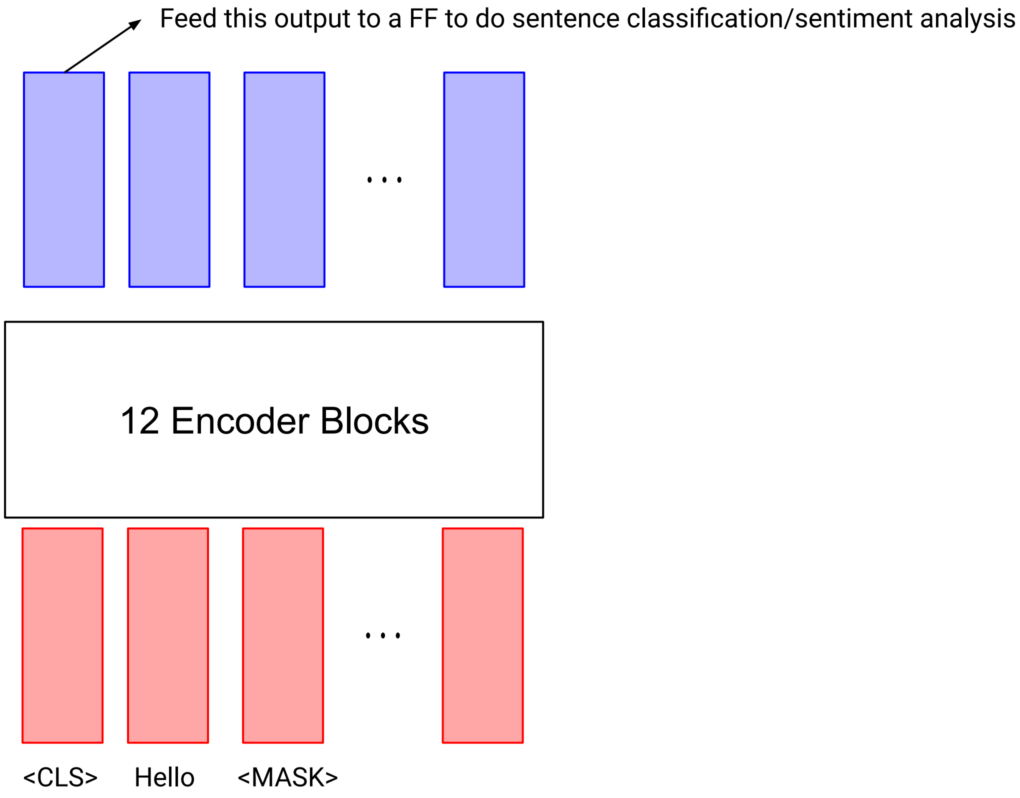
- <CLS> is a special token

Self-Supervised Learning

- BERT: Bidirectional Encoder Representations from Transformers
  - Encoder-only model
  - 12 Encoders
- Self-supervised Task #1

1. Take a mountain of text.
2. Mask ~15% of the words
3. Try to predict the masked words.

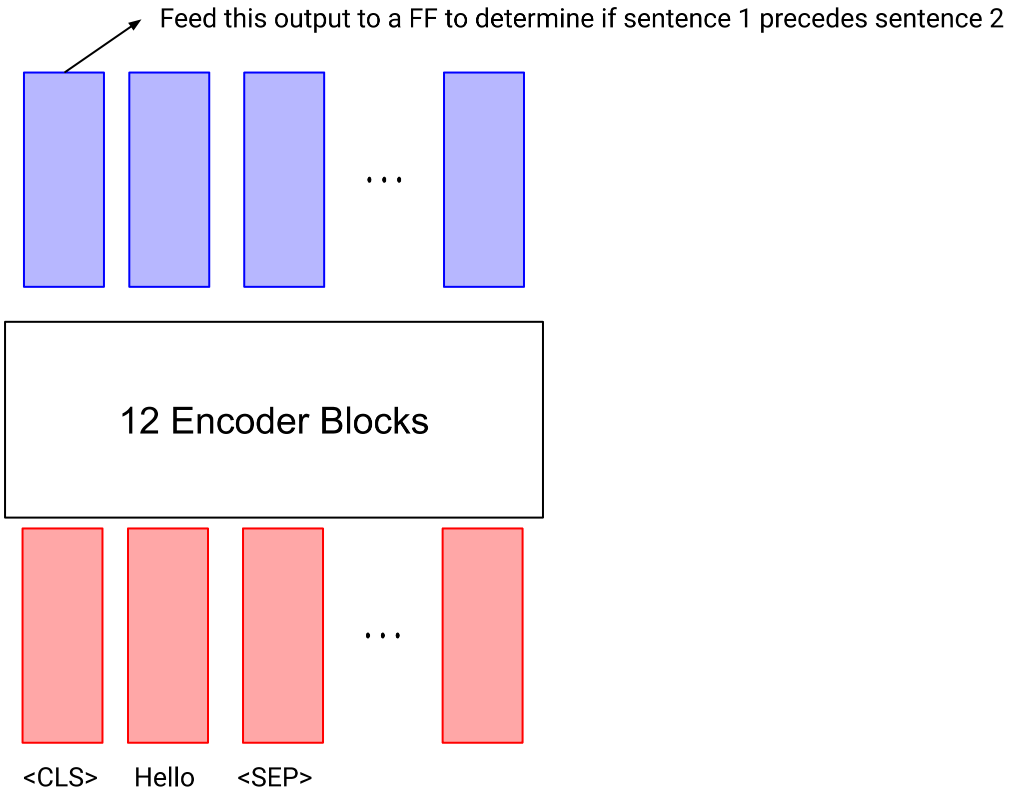
- <CLS> is a special token
- <SEP> is a token that separates sentences (Q/A tasks)



Self-Supervised Learning

- BERT: Bidirectional Encoder Representations from Transformers
  - Encoder-only model
  - 12 Encoders
- Self-supervised Task #2

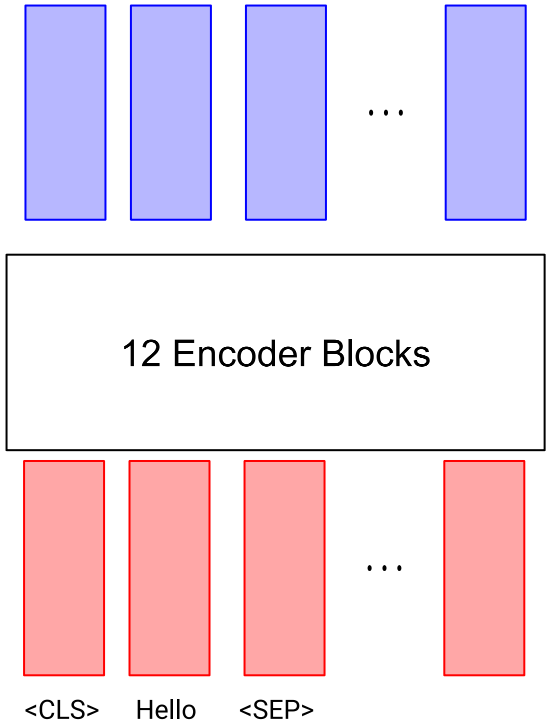
1. Take a mountain of text.
2. Take Two Sentences
3. Predict whether the first sentence follows the second



- <CLS> is a special token
- <SEP> is a token that separates sentences (Q/A tasks)

Self-Supervised Learning

- BERT: Bidirectional Encoder Representations from Transformers
  - Encoder-only model
  - 12 Encoders
- Fine-tune the pre-trained BERT for your task of choice!
- Can use frozen embeddings from any layer of BERT! (feature extraction)



Nice Blog Posts

- Illustrated Transformer
- Illustrated BERT