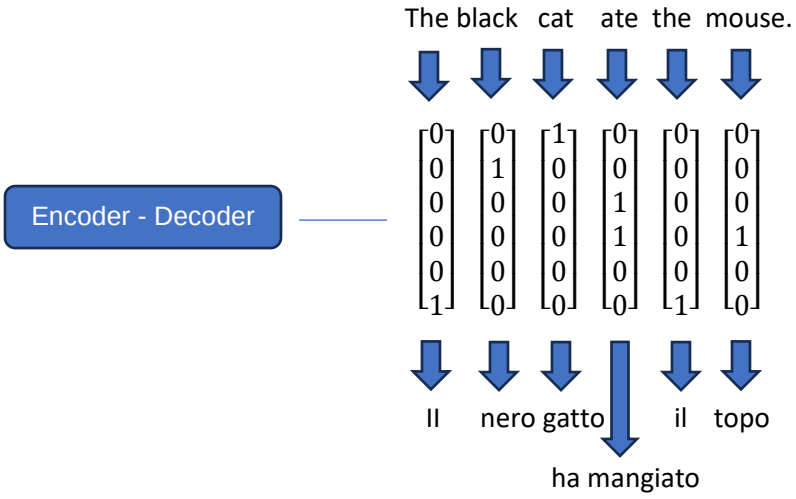
Lecture 4 Self-Attention: Attention Mechanism behind Transformer

1 Motivation

Encoder-decoder takes one word each time and translates from English to French. Sometimes, the words in source language do not align with the language in target language. Attention Mechanism is a technique that allows neural network to focus on a particular part of a sequence. It assigns weights to different parts of a sentence with the most important part receiving the highest weights.

Let's look at one example of word-to-word translation.



Word-to-Word translations made by Chat-GPT looks like the follows (which may carry errors):

Italian: "Il gatto nero ha mangiato il topo."

German: "Die schwarze Katze hat die Maus gefressen."

French: "Le chat noir a mangé la souris."

Spanish: "El gato negro se comió el ratón."

Japanese: "黒い猫がネズミを食べました。 (Kuroi neko ga nezumi o tabemashita.)"

Korean: 검은 고양이가 쥐를 먹었습니다. (Geomeun goyang-iga jwireul meogeossseubnida.)

Hindi: काली बिल्ली ने माउस को खाया। (Kālī billī ne mā'usa ko khāyā.)

Marathi: काळी मांजराने माउस खाला. (Kāḷī māñjarāne mā'usa khālā.)

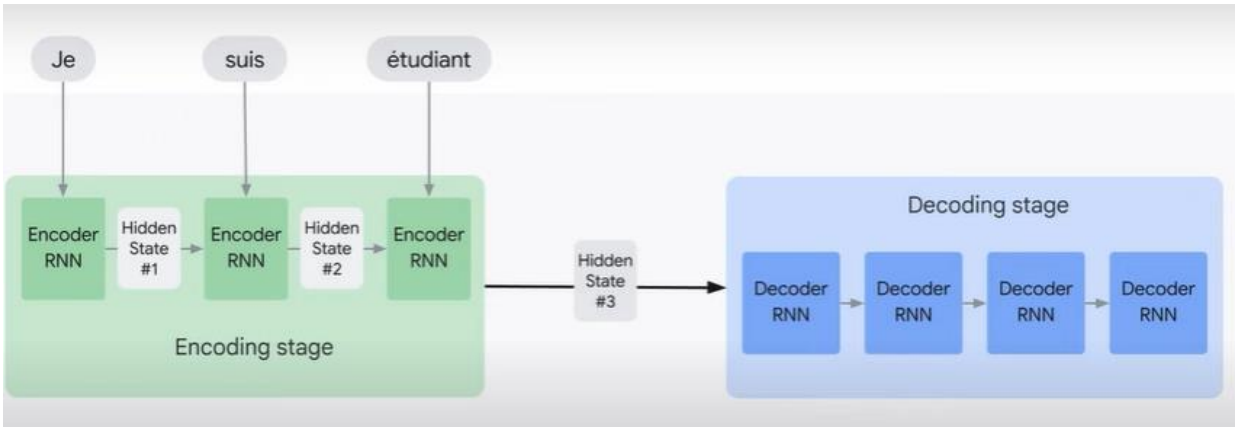
Kannada: ಕಪ್ಪು ಬೆಕ್ಕು ಎಲೆಹಿರುಳು ತಿಂದಿತು. (Kappu beku elehiruḷu tiṁdithu.)

Malayalam: കരിമനുഷ്യൻ ചൂണ്ടു കളിച്ചു. (Kari maṉuṣyaṉ cūṇṭu kaḷiccu.)

Gujarati: કાળી બિલાડી માઉસ ખાયું. (Kāḷī bilāḍī mā'usa khāyũ.)

Tamil: கருப்பு பூனை செல்லி சாப்பிட்டது. (Karuppu pūṉai celli sāppiṭṭatu.)

Traditional RNN encoder-decoder



(Je suis étudiant means "I am a student.")

RNN takes a particular part of a sentence $x(t)$ and passes it to the hidden state $h(t)$. In the end, only the final hidden state is passed on to the decoder. The decoder works with the hidden state for processing and translating it to the target language. The RNN forward pass can be represented by the below set of equations.

$$a(t) = b + W\, h(t-1) + Ux(t)$$

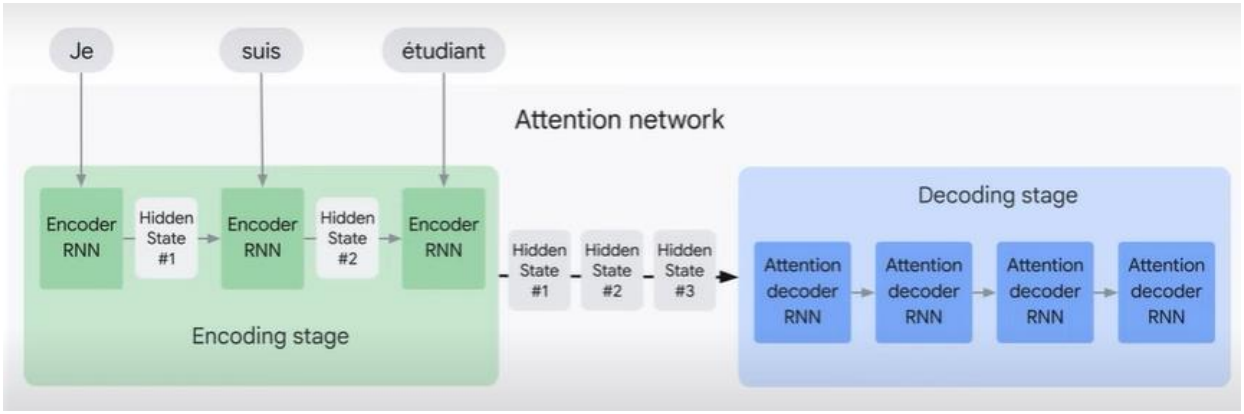
$$h(t) = \tanh(a(t))$$

$$o(t) = \mathrm{c} + Vh(t)$$

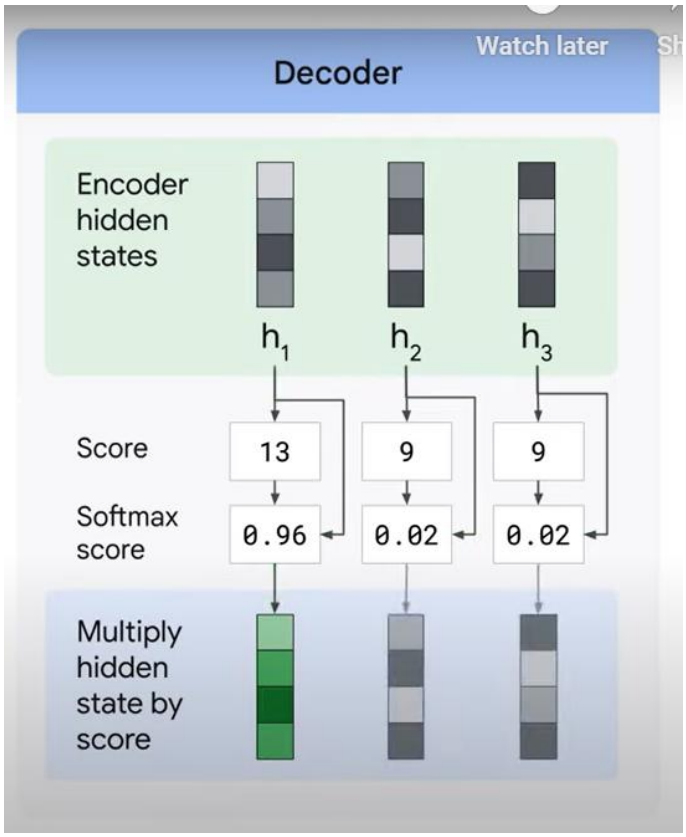
$$\hat{y}(t) = \mathrm{softmax}(\, o(t))$$

The Attention Mechanism Model differs from a traditional sequence to sequence model in two ways:

- First, the encoder passes a lot more data to the decoder. Instead of passing the last hidden state to the decoder, the Attention Model passes all hidden states to the decoder. This gives the decoder more context beyond the final hidden state. The decoder uses all the hidden states information to translate the sentence.

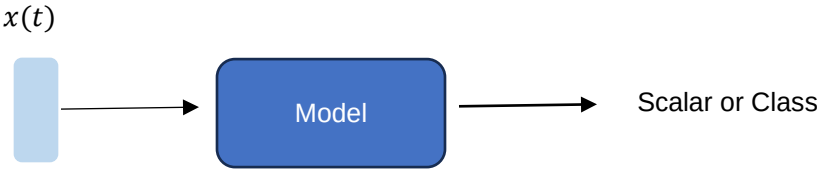


- Second, the Attention Mechanism adds an additional step to the decoder before producing the output. The Decoder does the following: First, it looks at the set of encoders hidden states it received. Each hidden state is associated with certain words from the input sentence. Second, it gives each hidden state a score. Third, it multiplies each hidden state by its soft-maxed score. This step magnifies the hidden state with higher scores but downsizes the hidden state with lower scores.

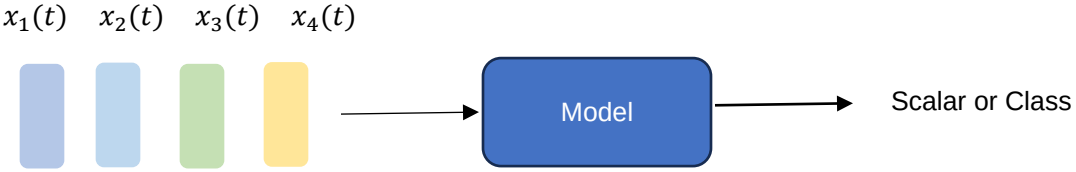


2 Design of Self Attention (Attention Mechanism)

RNN: At each timestamp t, the input is one vector.



Self-Attention Framework: At each timestamp t, the input is a sequence of vectors. The number of vectors is not fixed but may change from time to time.



For example, when the input is a sentence, the length of the sequence may vary from time to time.

This is a cat.

$x_1(t)$ $x_2(t)$ $x_3(t)$ $x_4(t)$

There are two methods that can transform words into vectors.

Method 1 to transform the word into vector is One-hot Encoding.

$$This = [0\ 0\ 1\ 0\ \dots]$$

$$is = [1\ 0\ 0\ 0\ \dots]$$

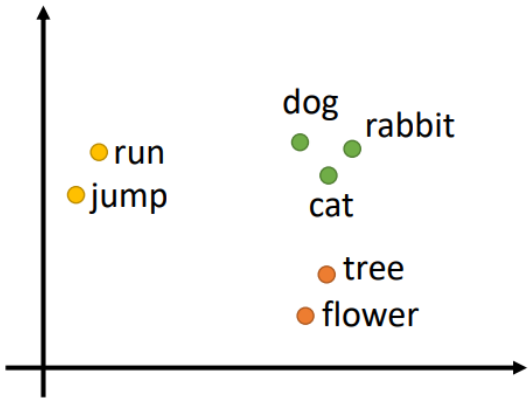
$$a = [0\ 2\ 0\ 0\ \dots]$$

$$cat = [0\ 0\ 0\ 1\ \dots]$$

The issue here is that the above vectorization process simply assumes there is NO correlation among these words.

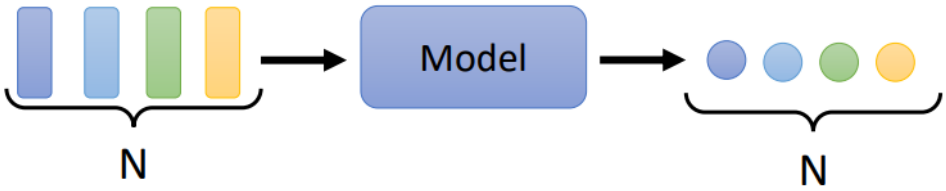
Method 2 to transform the word into vector is Word Embedding.

When you visualize the vectorized words as below, you may find all animals are agglomerated together, all plants are agglomerated together, and all verbs are agglomerated together. One sentence is a set of vectors with various lengths.



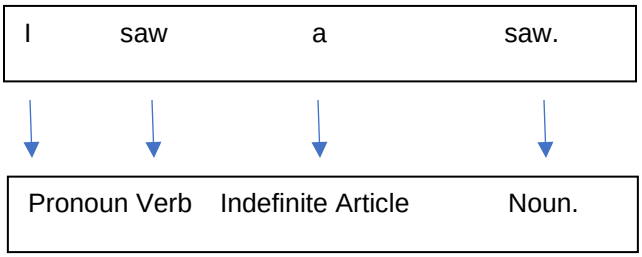
Output Types

Type 1 Each vector has a label. If the input is the size of 4, the output is the size of 4. If the label is a numerical value, the model is a regression model. If the label is a class, the model is a classification model. The model doesn't determine the output size. The output size is the same as the input.



Example: POS Tagging

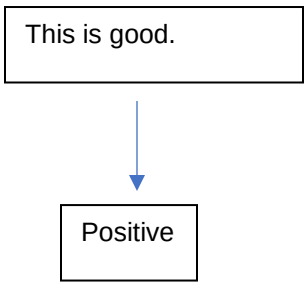
In corpus linguistics, part-of-speech tagging (POS tagging or PoS tagging or POST), also called grammatical tagging is the process of marking up a word in a text (corpus) as corresponding to a particular part of speech, based on both its definition and its context. The model needs to determine the part of speech for each word.



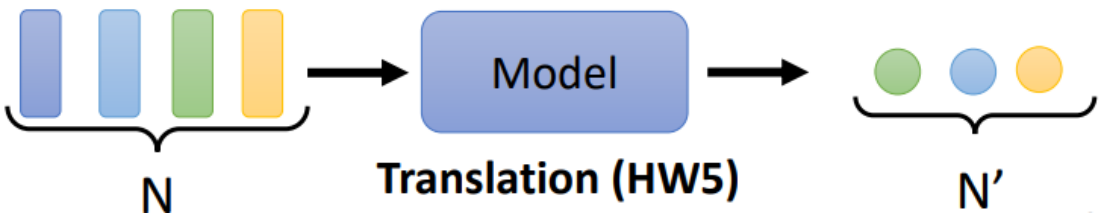
Type 2: There is one label for the whole output sequence.

Example: Sentiment Analysis

Sentiment Analysis

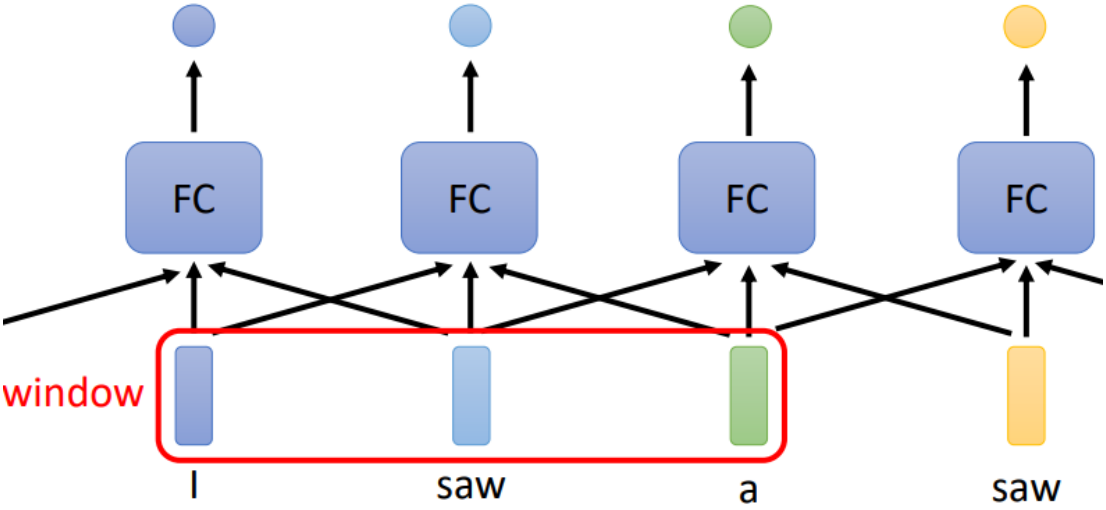


Type 3: Model decides the number of labels itself. This type of task is so-called 'Seq2Seq'(sequence-to-sequence).



3 Data Preparation

Let's use Type 1 model to illustrate how self-attention works. The 2nd word and the 4th word have different structures and meaning. But the fully connected network (FC) is configured to return the same output give the same input. How does the fully connected network (FC) differentiate between the 2nd word and the 4th word? How to let the fully connected network (FC) consider the context of input?

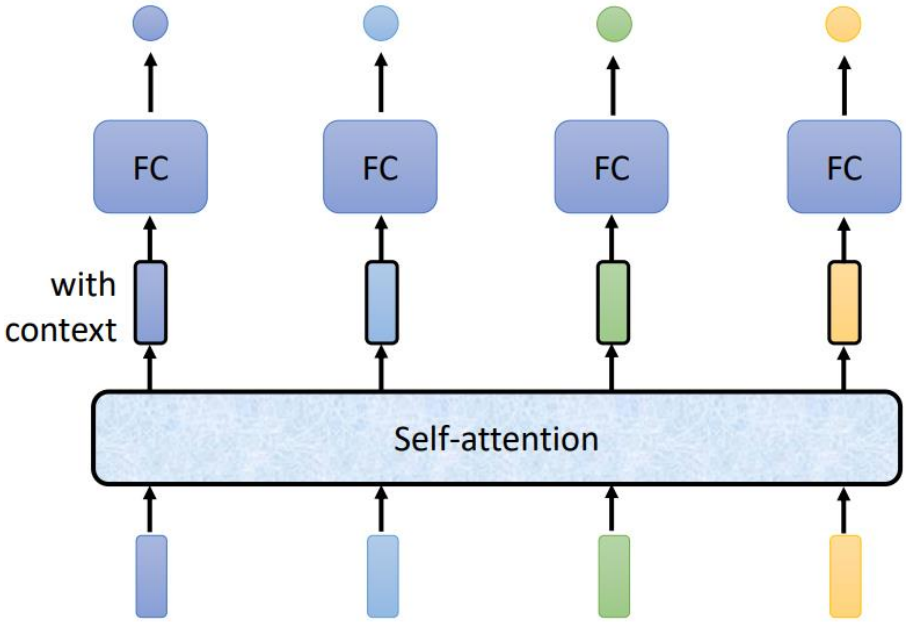


Solution 1: we add a window to connect the adjacent words.

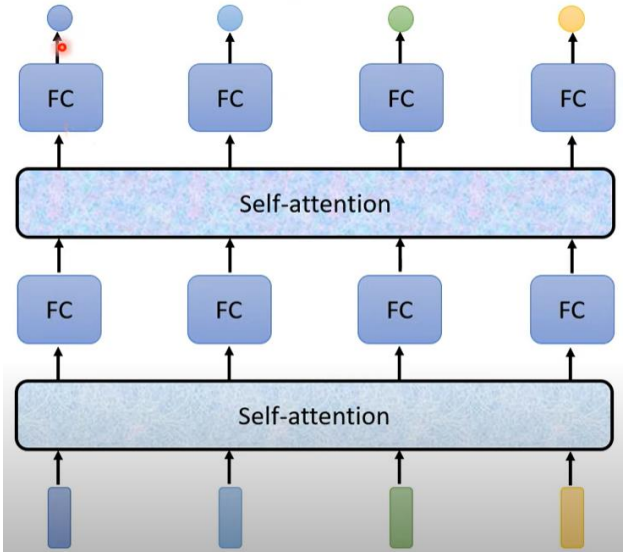
Solution 2: we connect the words in the entire input sequence using a large window.

It is noteworthy that the length of the sequence may vary from time to time. Using a window with the fixed size may not work for every input. In the meantime, a large window requires the FC to use more parameters, which is not only computationally expensive, but also may lead to over-fitting problem.

The self-attention mechanism is designed to take account of the context of the entire input sequence. Self-attention will output the same number of outputs as the inputs. If inputs are composed of four vectors, the output (from self-attention) will contain four vectors too. After applying self-attention, the output from self-attention feeds into FC. The black boarder signifies the output from self-attention takes into account of the context of the entire input sequence.

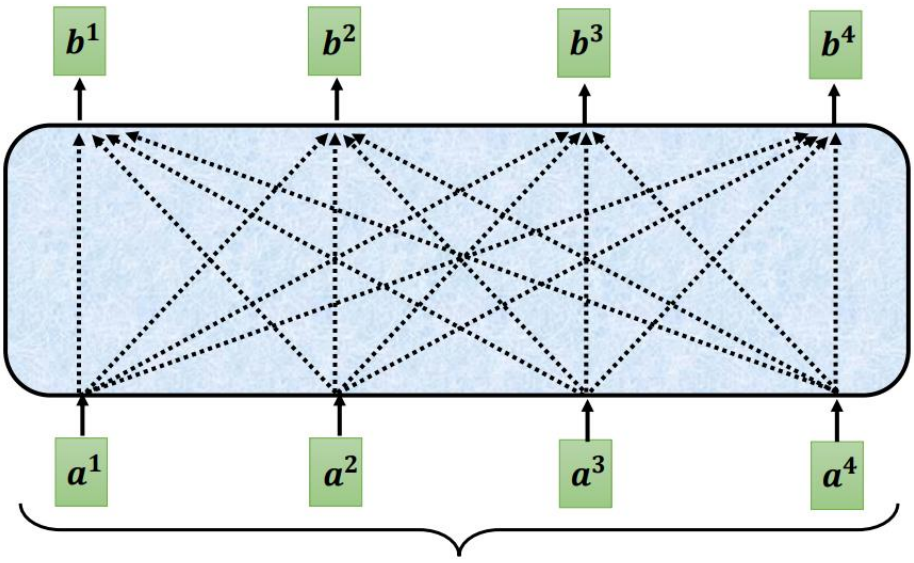


In fact, you could iteratively use the self-attention layer and FC layer several times to fully incorporate the context of the input sequence. FC focuses on processing the local information on a particular part of the input sequence. Self-attention aims to deal with the context of the entire input sequence. In the famous paper published by Google researchers: "Attention is all you need" (https://arxiv.org/abs/1706.03762), the self-attention is the key structure for Transformer Models.



4 Self-Attention Model Configuration

The self-attention can take either the raw input or the output from a hidden layer as input. In the diagram below.



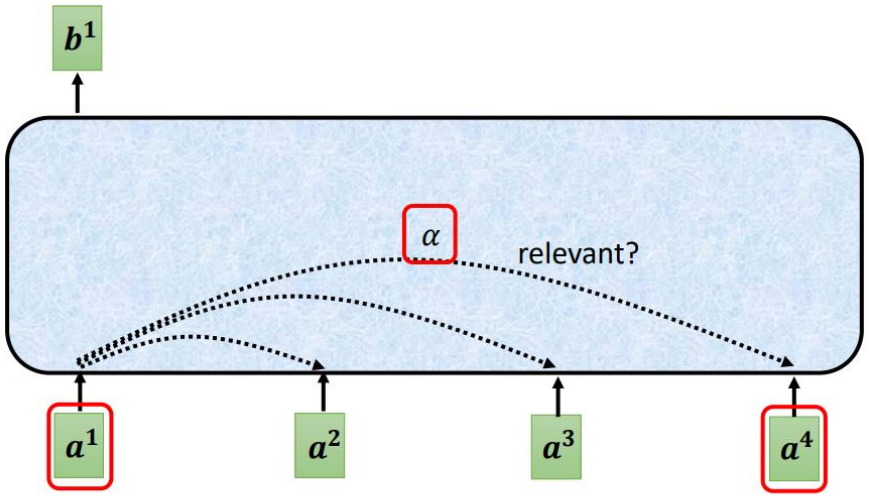
a^1, a^2, a^3, a^4 are input vectors fed into self-attention. We use a^i but not x^i to represent the input for self-attention because a^i may be the intermediate output from a particular layer and thus pre-processed x^i.

b^1, b^2, b^3, b^4 are output vectors obtained from self-attention. Each b^i considers all input vectors (the entire input sequence): a^1, a^2, a^3, a^4.

Let's take b^1 as the example and see how b^i are obtained by self-attention.

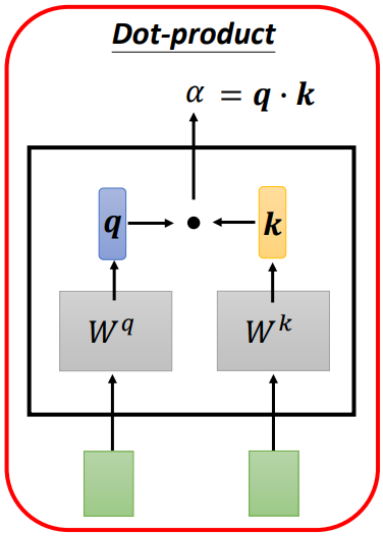
Step 1: Methods illustration - how to get the attention score α.

The purpose of self-attention is to find relevant vectors in a sequence. Given a^1, which among a^2, a^3, a^4 are relevant to determine the context of a^1. In the following example, we use a scalar α to represent the relevance (not correlation) between a^1 and a^4. The scalar α is formally named "attention score."



Method 1: Dot Product

Here, the input a^1 is a $M \times 1$ vector.



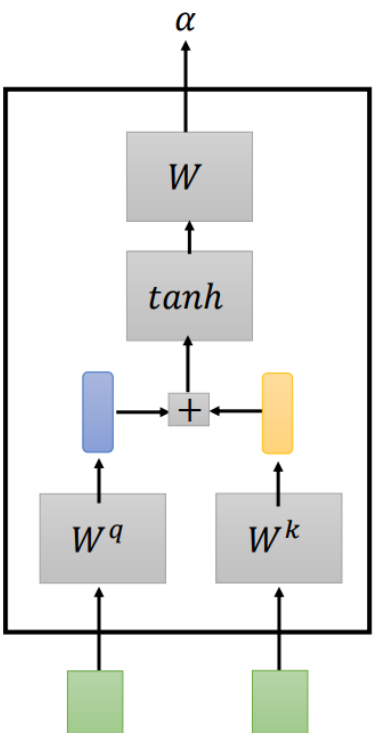
$$q = W^q a^1$$

$$k = W^k a^1$$

$$a = q \cdot k$$

- q represents query.
- k represents the key.

Method 2: Additive



$$q = W^q a^1$$

$$k = W^k a^1$$

$$\tilde{\alpha} = \tanh(q + k)$$

$$a = W \cdot \tilde{\alpha}$$

Step 2: Compute the pair-wise ***Attention Score*** between a^1 and a^2, a^3, a^4 as follows.

We first calculate the query of a^1 by

$$q^1 = W^q a^1$$

And the keys of a^2, a^3, a^4 by

$$k^2 = W^k a^2$$

$$k^3 = W^k a^3$$

$$k^4 = W^k a^4$$

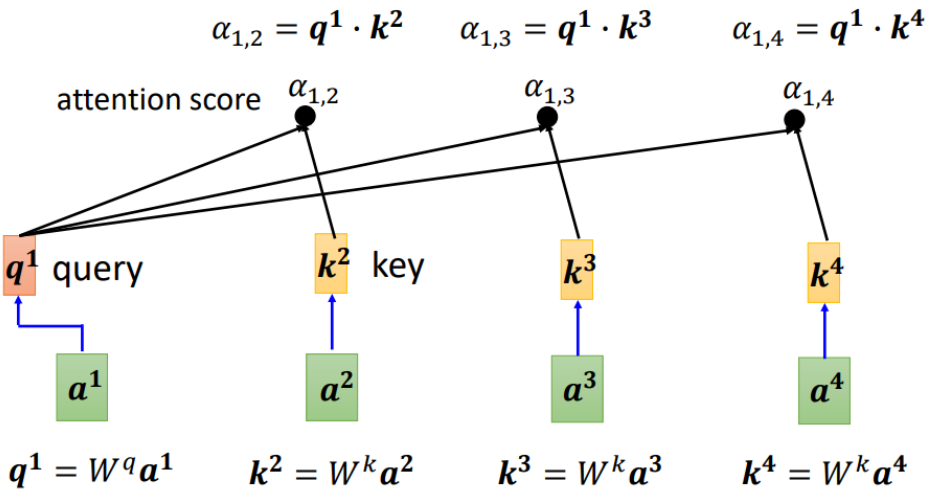
The relevance score between a^1 and a^2, a^3, a^4 are:

$$\alpha_{1,2} = q^1 k^2$$

$$\alpha_{1,3} = q^1 k^3$$

$$\alpha_{1,4} = q^1 k^4$$

The calculation process can be illustrated as follows.



In addition, we need to calculate the attention score of a^1 with itself.

$$q^1 = W^q a^1$$

$$k^1 = W^k a^1$$

$$\alpha_{1,1} = q^1 \cdot k^1$$

All attention scores then need to go through a soft-max process to obtain:

$$\alpha'_{1,1} = softmax\left(\alpha_{1,1}\right) = \frac{e^{\alpha_{1,1}}}{e^{\alpha_{1,1}} + e^{\alpha_{1,2}} + e^{\alpha_{1,3}} + e^{\alpha_{1,4}}}$$

$$\alpha'_{1,2} = softmax\left(\alpha_{1,2}\right) = \frac{e^{\alpha_{1,2}}}{e^{\alpha_{1,1}} + e^{\alpha_{1,2}} + e^{\alpha_{1,3}} + e^{\alpha_{1,4}}}$$

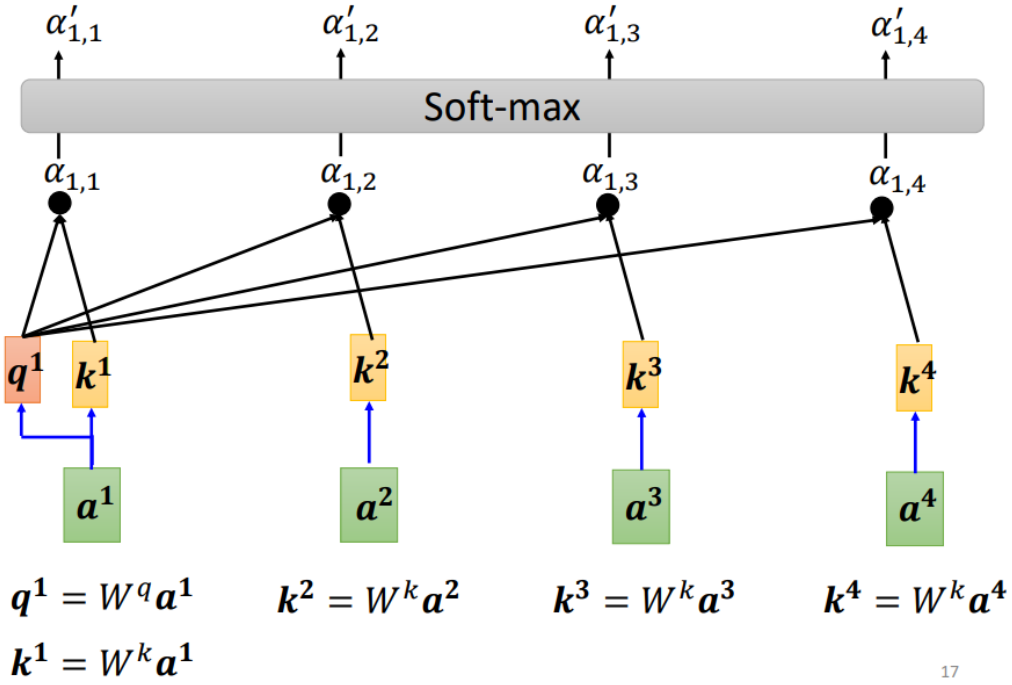
$$\alpha'_{1,3} = softmax\left(\alpha_{1,3}\right) = \frac{e^{\alpha_{1,3}}}{e^{\alpha_{1,1}} + e^{\alpha_{1,2}} + e^{\alpha_{1,3}} + e^{\alpha_{1,4}}}$$

$$\alpha'_{1,4} = softmax\left(\alpha_{1,1}\right) = \frac{e^{\alpha_{1,4}}}{e^{\alpha_{1,1}} + e^{\alpha_{1,2}} + e^{\alpha_{1,3}} + e^{\alpha_{1,4}}}$$

In all, we could summarize the softmax process as:

$$\alpha'_{1,i} = \frac{exp(\alpha_{1,i})}{\sum_j^N exp\left(\alpha_{1,j}\right)}$$

You don't have to use softmax. You have full freedom to choose any scaling functions, e.g. ReLU.



Step 3: Extract the meaning information using the attention scores.

For each input a^1 and a^2, a^3, a^4, we need to calculate the new vector v^i, which represents the meaning:

$$v^1 = W^v a^1$$

$$v^2 = W^v a^2$$

$$v^3 = W^v a^3$$

$$v^4 = W^v a^4$$

We then multiply the above v^i vectors with the relevance scores. The output b^i is then obtained by summing up these products.

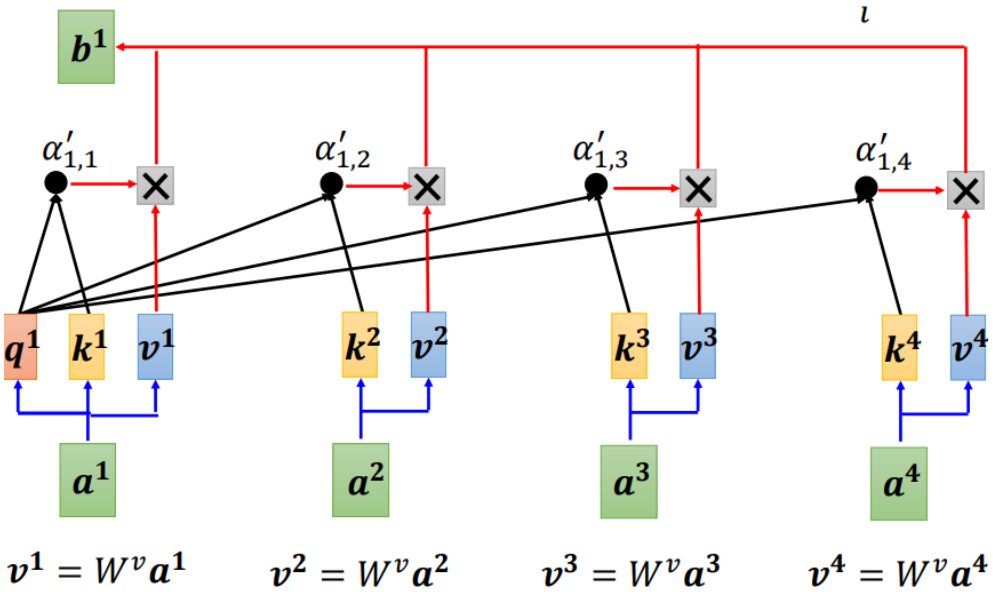
$$b^1 = \alpha'_{1,1} v^1 + \alpha'_{1,2} v^2 + \alpha'_{1,3} v^3 + \alpha'_{1,4} v^4$$

we could summarize the calculation as:

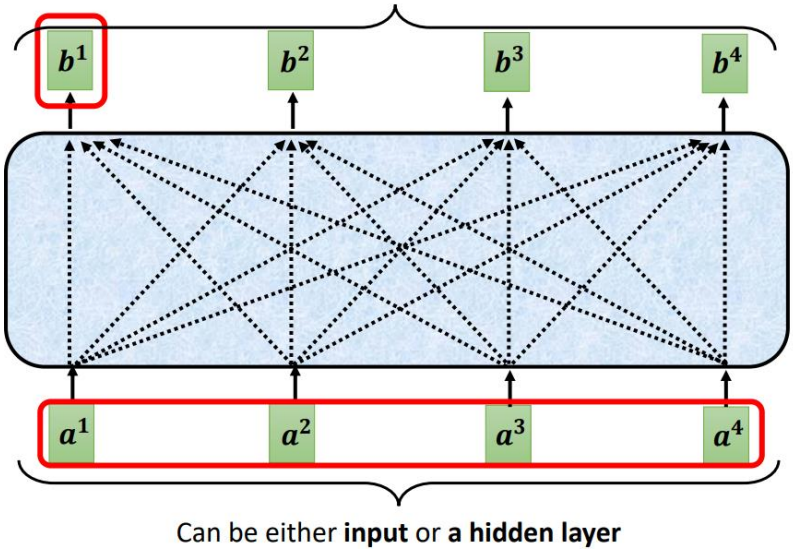
$$b^1 = \sum_{i=1}^{K} \alpha'_{1,i} \cdot v^i$$

Here the attention score works like a scalar. If an input a^i is highly relevant to a^1 , it should receive a high relevance score $\alpha'_{1,i}$. The high relevance score will help v^i dominate the output b^1. The length of the input sequence is K, with each input a $M \times 1$ vector.

The above process is illustrated in the diagram below.



You can parallel the above process to obtain b^1, b^2, b^3, b^4 respectively.



Let's repeat the above process to calculate b^2. In practice, you could parallel the calculation of b^i for $i = N$.

$$q^2 = W^q a^2$$

$$k^2 = W^k a^2$$

Next:

$$\alpha_{2,1} = q^2 \cdot k^1$$

$$\alpha_{2,2} = q^2 \cdot k^2$$

$$\alpha_{2,3} = q^2 \cdot k^3$$

$$\alpha_{2,4} = q^2 \cdot k^4$$

It is noteworthy that N defines the range the self-attention mechanism is attached to. In speech recognition, you may not need the full sentence but only a range around each work input. You could use $N < T$ to calculate the attention scores.

Normalization or softmax to obtain the attention score (relevance score):

$$\alpha'_{2,1} = softmax\left(\alpha_{2,1}\right) = \frac{e^{\alpha_{2,1}}}{e^{\alpha_{2,1}} + e^{\alpha_{2,2}} + e^{\alpha_{2,3}} + e^{\alpha_{2,4}}}$$

$$\alpha'_{2,2} = softmax\left(\alpha_{2,2}\right) = \frac{e^{\alpha_{2,2}}}{e^{\alpha_{2,1}} + e^{\alpha_{2,2}} + e^{\alpha_{2,3}} + e^{\alpha_{2,4}}}$$

$$\alpha'_{2,3} = softmax\left(\alpha_{2,3}\right) = \frac{e^{\alpha_{2,3}}}{e^{\alpha_{2,1}} + e^{\alpha_{2,2}} + e^{\alpha_{2,3}} + e^{\alpha_{2,4}}}$$

$$\alpha'_{2,4} = softmax\left(\alpha_{2,4}\right) = \frac{e^{\alpha_{2,4}}}{e^{\alpha_{2,1}} + e^{\alpha_{2,2}} + e^{\alpha_{2,3}} + e^{\alpha_{2,4}}}$$

Then we are ready to calculate b^2. v^1, v^2, v^3 and v^4 are the same as those used in calculating b^1.

$$b^2 = \alpha'_{2,1} v^1 + \alpha'_{2,2} v^2 + \alpha'_{2,3} v^3 + \alpha'_{2,4} v^4$$

we could summarize the calculation as:

$$b^2 = \sum_{i=1}^{K} \alpha'_{2,i} \cdot v^i$$

Up to here, you may realize that all input a^1, a^2, a^3 and a^4 share the same parameters W^q, W^k, W^v to obtain q, k and v. Given the fact that each input a^i is a $M \times 1$ vector, we could stack all inputs into one $M \times K$ matrix.

$$[q^1 \quad q^2 \quad q^3 \quad q^4] = W^q \cdot [a^1 \quad a^2 \quad a^3 \quad a^4]$$

$$[k^1 \quad k^2 \quad k^3 \quad k^4] = W^k \cdot [a^1 \quad a^2 \quad a^3 \quad a^4]$$

$$[v^1 \quad v^2 \quad v^3 \quad v^4] = W^v \cdot [a^1 \quad a^2 \quad a^3 \quad a^4]$$

Let $I = [a^1 \quad a^2 \quad a^3 \quad a^4]_{M \times K}$. Parameters of the network are:

$$W^q: 1 \times M$$

$$W^k: 1 \times M$$

$$W^v: 1 \times M$$

Calculate the inner product of q and k to obtain attention score.

$$\begin{bmatrix} \alpha_{1,1} \\ \alpha_{1,2} \\ \alpha_{1,3} \\ \alpha_{1,4} \end{bmatrix} = \begin{bmatrix} k^1 \\ k^2 \\ k^3 \\ k^4 \end{bmatrix} q^1$$

$$\begin{bmatrix} \alpha_{2,1} \\ \alpha_{2,2} \\ \alpha_{2,3} \\ \alpha_{2,4} \end{bmatrix} = \begin{bmatrix} k^1 \\ k^2 \\ k^3 \\ k^4 \end{bmatrix} q^2$$

which is to say,

$$\begin{bmatrix} \alpha_{1,1} & \alpha_{2,1} & \alpha_{3,1} & \alpha_{4,1} \\ \alpha_{1,2} & \alpha_{2,2} & \alpha_{3,3} & \alpha_{4,2} \\ \alpha_{1,3} & \alpha_{2,3} & \alpha_{3,3} & \alpha_{4,3} \\ \alpha_{1,4} & \alpha_{2,4} & \alpha_{3,4} & \alpha_{4,4} \end{bmatrix} = \begin{bmatrix} k^1 \\ k^2 \\ k^3 \\ k^4 \end{bmatrix} \cdot [q^1 \quad q^2 \quad q^3 \quad q^4]$$

i.e.

$$A = K^T \cdot Q$$

Taking softmax, the sum of each column must be 1.

$$\begin{bmatrix} \alpha'_{1,1} & \alpha'_{2,1} & \alpha'_{3,1} & \alpha'_{4,1} \\ \alpha'_{1,2} & \alpha'_{2,2} & \alpha'_{3,3} & \alpha'_{4,2} \\ \alpha'_{1,3} & \alpha'_{2,3} & \alpha'_{3,3} & \alpha'_{4,3} \\ \alpha'_{1,4} & \alpha'_{2,4} & \alpha'_{3,4} & \alpha'_{4,4} \end{bmatrix} = softmax\left\{ \begin{bmatrix} \alpha_{1,1} & \alpha_{2,1} & \alpha_{3,1} & \alpha_{4,1} \\ \alpha_{1,2} & \alpha_{2,2} & \alpha_{3,3} & \alpha_{4,2} \\ \alpha_{1,3} & \alpha_{2,3} & \alpha_{3,3} & \alpha_{4,3} \\ \alpha_{1,4} & \alpha_{2,4} & \alpha_{3,4} & \alpha_{4,4} \end{bmatrix} \right\}$$

To calculate $[b^1 \quad b^2 \quad b^3 \quad b^4]$, we need

$$[b^1 \quad b^2 \quad b^3 \quad b^4] = [v^1 \quad v^2 \quad v^3 \quad v^4] \begin{bmatrix} \alpha'_{1,1} & \alpha'_{2,1} & \alpha'_{3,1} & \alpha'_{4,1} \\ \alpha'_{1,2} & \alpha'_{2,2} & \alpha'_{3,3} & \alpha'_{4,2} \\ \alpha'_{1,3} & \alpha'_{2,3} & \alpha'_{3,3} & \alpha'_{4,3} \\ \alpha'_{1,4} & \alpha'_{2,4} & \alpha'_{3,4} & \alpha'_{4,4} \end{bmatrix}$$

i.e.

$$O = V \cdot A'$$

To summarize the above procedure,

$$Q = W^q \cdot I$$

$$K = W^k \cdot I$$

$$V = W^v \cdot I$$

Next:

$$A = K^T \cdot Q$$

The attention matrix is:

$$A' = softmax\{A\}$$

The output is:

$$O = V \cdot A'$$

The input is I. The output is O. The parameters to be estimated (learned) are: W^q, W^k, W^v.

Multi-head Self-attention

Multi-head self-attention now finds wide applications Generative AI apps. Tasks like translation or speech recognition need to use more heads, . Take 2-heads self-attention as example, there might be different types of relevance. Using one q cannot capture all of them, we need to introduce multiple qs into our model. Specifically, we take the following steps to incorporate multi-heads into our model.

The first step to obtain $q^i = W^q a^i$ now can be decomposed into two-heads.

$$q^{i,1} = W^{q,1} a^i$$

$$q^{i,2} = W^{q,2} a^i$$

$q^{i,1}$ and $q^{i,2}$ represent two types of relevance. Consequently, there have to be $k^{i,1}$, $k^{i,2}$ and $v^{i,1}$, $v^{i,2}$ to match $q^{i,1}$, $q^{i,2}$.

Next the calculation of self-attention score takes place along each head:

$$\alpha_{i,i,1} = q^{i,1} \cdot k^{i,1}$$

$$\alpha_{i,j,1} = q^{i,1} \cdot k^{j,1}$$

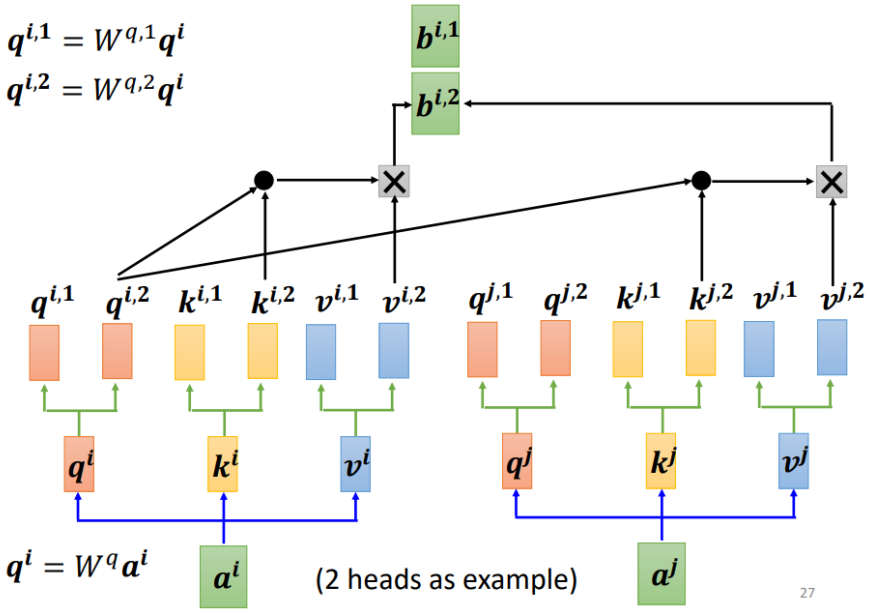
$$\alpha'_{i,i,1} = softmax(\alpha_{i,i,1})$$

$$\alpha'_{i,j,1} = softmax(\alpha_{i,j,1}), \quad j = 2,3,4$$

$$b^{i,1} = \alpha'_{i,i,1} \cdot v^{i,1} + \sum_{j=2}^{4} \alpha'_{i,j,1} \cdot v^{j,1}$$

...

The above procedure could be illustrated as follows.



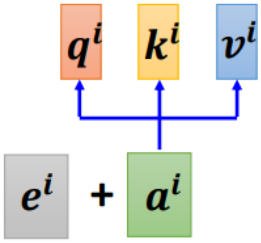
Finally

$$b^i = W^O \cdot \begin{bmatrix} b^{i,1} \\ b^{i,2} \end{bmatrix}$$

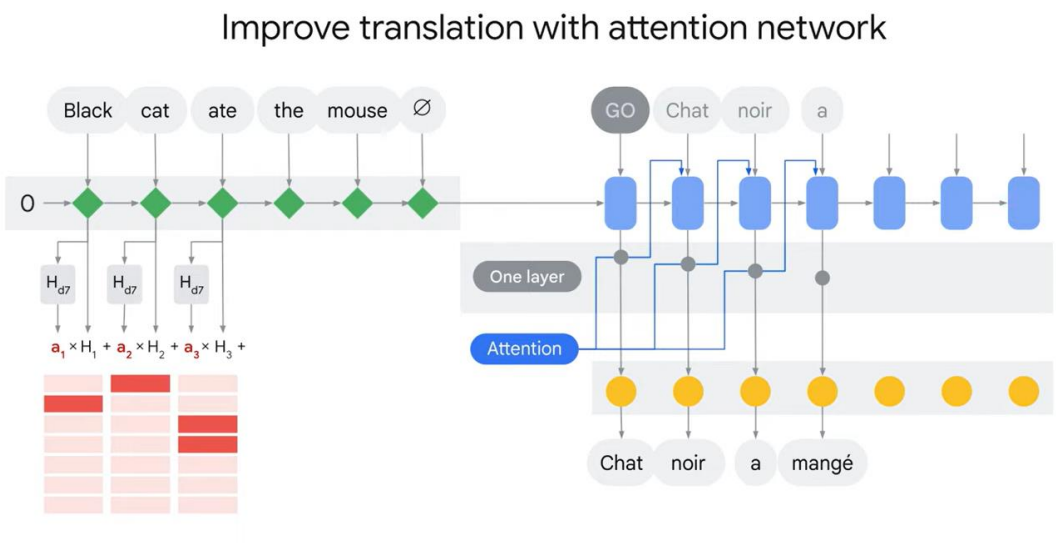
Positional Encoding

No position information has been included in the self-attention above. How to incorporate positional information? Use positional encoding. In particular,

Each position has a unique positional vector e^i. Different positions are labeled with different positional vectors. Input at each position is simply updated by adding e^i on top of input a^i. The positional vectors e^i are manually specified and often obtained using $\sin(\cdot)\ and\ \cos(\cdot)$ functions.



5 Application: Improve Translation with Attention Networks



α represents the attention rate at each time step. $H_{d,t}$ represents the hidden state of the decoder RNN at each time step. With the attention mechanism, the inversion of the Black Cat translation is clearly visible in the attention diagram and "ate" translates as two words, "a mange", in French. We can see the attention network staying focused on the word "ate" for two time steps. During the attention step we use the encoder hidden states and the H_4 vector to calculate a context vector a_4 for this time step. This is the weighted sum. We then concatenate H_4 and α_4 into one vector, $\alpha_4 \times H_4$. This concatenated vector is passed through a feedforward neural network. One train jointly with the model to predict the next work. The output of the feedforward neural network indicates the output word of this time step. This process continues till the end of sentence token is generated by the decoder. This is how you can use an attention mechanism to improve the performance of a traditional encoder decoder architecture.

Appendix

A1 Encoder and Decoder

In the context of deep learning, "encoder" and "decoder" refer to components of certain types of neural network architectures, particularly in the realm of autoencoders and sequence-to-sequence models.

Encoder

- In the context of autoencoders, the encoder is a neural network that takes an input (such as an image, text, or some other data representation) and compresses it into a latent space representation, often of lower dimensionality than the input.
- In sequence-to-sequence models like those used in machine translation or text summarization, the encoder processes the input sequence (e.g., a sentence in one language) and transforms it into a fixed-size context vector or a sequence of vectors that captures the meaning or context of the input.

Decoder

- In autoencoders, the decoder is a neural network that takes the compressed representation produced by the encoder and attempts to reconstruct the original input from it.
- In sequence-to-sequence models, the decoder takes the context vector(s) produced by the encoder and generates an output sequence (e.g., a translation in another language or a summary of the input text).

The encoder and decoder are typically trained jointly, with the objective of minimizing some form of reconstruction error in autoencoders or maximizing the likelihood of generating the correct output sequence in sequence-to-sequence models. These components are fundamental in various deep learning tasks such as image denoising, representation learning, machine translation, and more.

Reference

https://speech.ee.ntu.edu.tw/~hylee/ml/ml2021-course-data/self_v7.pdf
https://www.youtube.com/watch?v=hYdO9CscNes
https://arxiv.org/abs/1706.03762