Lecture 4 Attention Mechanisms

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.

English	Encoder - Decoder	**French**	**Italy**
The	[0 0 0 0 1]	le	Il
black	[0 1 0 0 0]	noir	nero
cat	[1 0 0 0 0]	chat	gatto
ate	[0 0 1 1 0]	a mangé	ha mangiato
the	[0 0 0 0 1]	la	il
mouse	[0 0 1 0 0]	souris	topo

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."

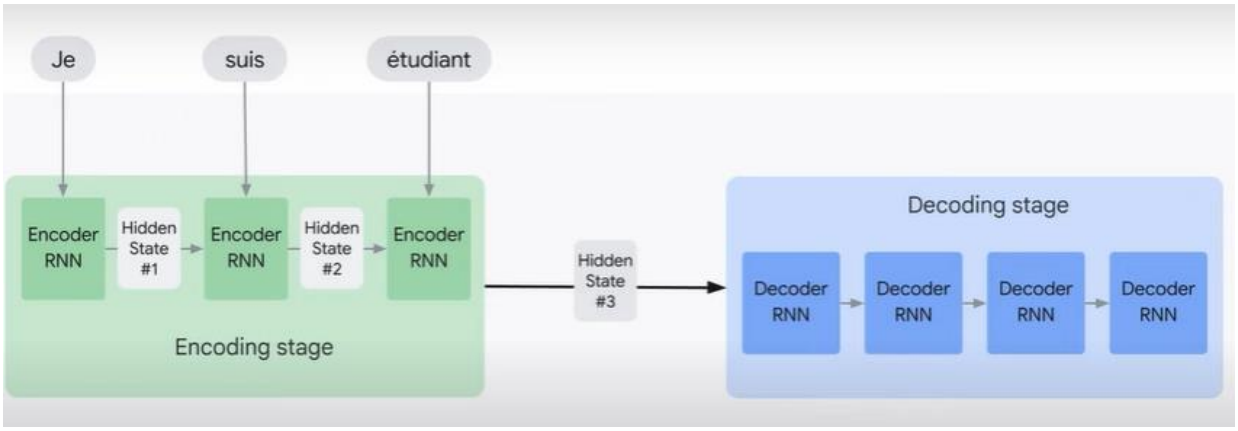
Spanish: "El gato negro se comió el ratón."

Chinese: 黑猫吃老鼠。

Hindi: काले बिल्ली ने चूहे को खा लिया।

Analyze the sentence in Marathi, Kannada, Gujarati and Tamil now.

Traditional RNN encoder-decoder



RNN takes each word sequentially in a sentence $x(t)$ and passes it to its own hidden state $h(t)$ interactively. 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.

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

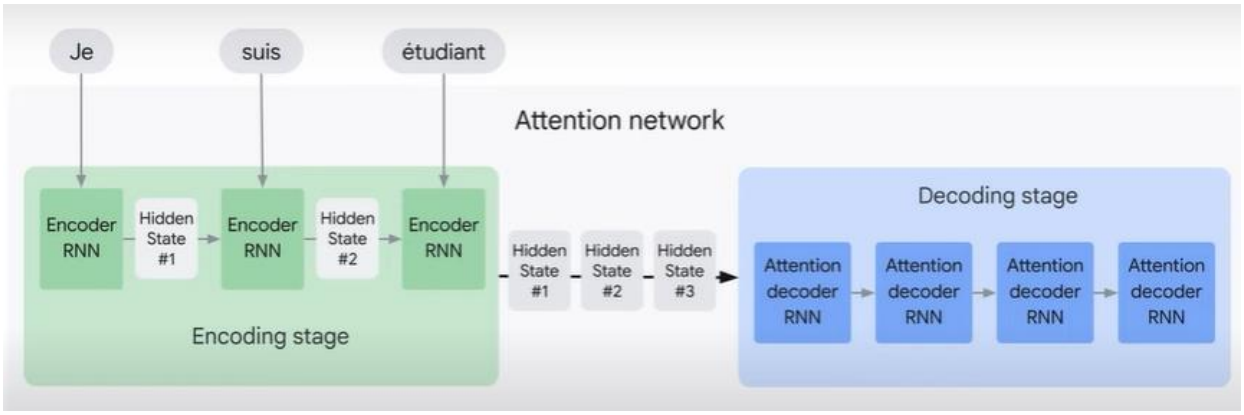
$$h(t) = \tanh(a(t))$$

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

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

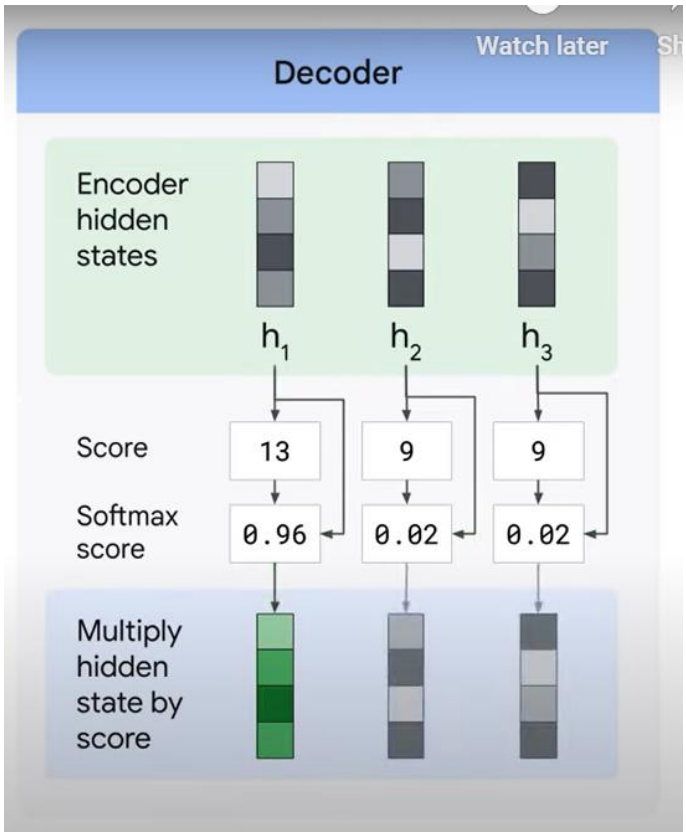
The Attention Mechanism Model differs from a traditional sequence to sequence model in two ways:

- First, the encoder passes data simultaneously into the decoder. Instead of passing the hidden state from the last word 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-max score. This

step magnifies the hidden state with higher scores but downsizes the hidden state with lower scores.



2 Types of Attention Mechanism

Attention mechanisms are widely used in machine learning, particularly in natural language processing (NLP), computer vision, and other deep learning tasks. The idea behind attention mechanisms is to focus on specific parts of the input data while processing, which helps improve performance by emphasizing relevant information. Here are the main types of attention mechanisms:

1. Soft Attention

- **Description**: Soft attention assigns different weights to different parts of the input. The model then computes a weighted sum of the input elements.
- **Use Case**: This type of attention is differentiable, which makes it suitable for end-to-end training with gradient-based optimization.
- **Common Applications**: NLP tasks like machine translation, text summarization, and image captioning.

2. Hard Attention

- **Description**: Unlike soft attention, hard attention selects one part of the input with a binary choice (either attend or not attend). It is non-differentiable, and often requires reinforcement learning to train.
- **Use Case**: Hard attention is computationally more efficient than soft attention but harder to train.

- **Common Applications**: Scenarios where efficiency is more important than differentiability, like certain computer vision tasks.

3. Self-Attention (Intra-Attention)

- **Description**: Self-attention computes the relevance of each input token with respect to all other tokens in the same sequence. It is particularly useful in capturing dependencies between words or elements in a sequence.
- **Use Case**: Self-attention is the core of models like the Transformer, where it allows for more parallelization.
- **Common Applications**: Language models (e.g., GPT, BERT), machine translation, and text generation.

4. Multi-Head Attention

- **Description**: Multi-head attention allows the model to focus on different parts of the input simultaneously by having multiple "heads" compute attention scores in parallel. Each head captures different features or relationships.
- **Use Case**: This is a key component of the Transformer model, providing the ability to look at multiple aspects of the input.
- **Common Applications**: Used extensively in models like BERT, GPT, and Vision Transformers (ViTs).

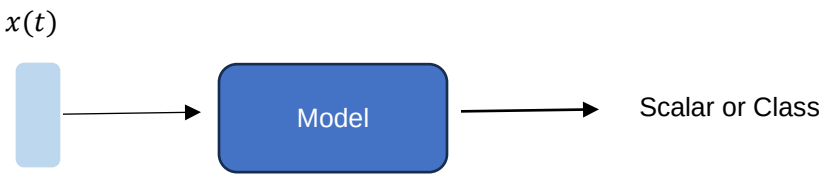
5. Cross-Attention

- **Description**: Cross-attention involves one set of queries, keys, and values attending to another set. This is common in models that process two different inputs, like an image and text, and require information from both.
- **Use Case**: When two different inputs need to attend to each other, such as in tasks combining text and images or in multi-modal applications.
- **Common Applications**: Multi-modal tasks (e.g., vision-language models), machine translation, text generation tasks.

Each type of attention mechanism is suited for different tasks and model architectures, depending on the complexity and requirements of the application.

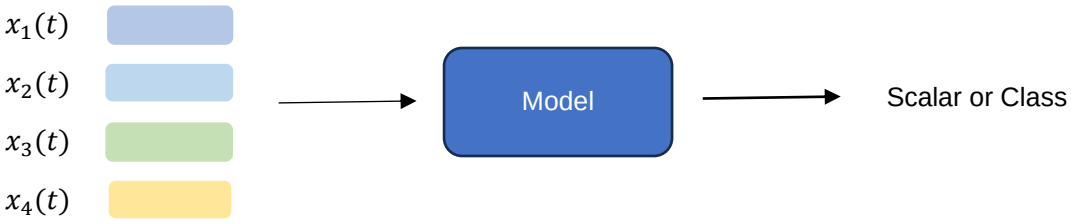
3 Design of Self Attention (Attention Mechanism)

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

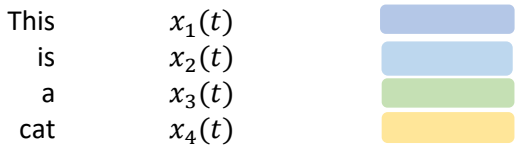


Input Types

At each timestamp t, the input is a sequence of vectors. Though the size of each vector is fixed, 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. We simply denote the length of the sequence to be N, which is usually set to be the maximum length of the sentence. The length of the embedding vector to represent each word depends on the size of the vocabulary used in the training data, which is usually denoted as d_{model} or simply d.



There are two methods that can transform words into vectors.

Method 1 to transform the word into vector by One-hot Encoding. Oxford English Corpus (OEC) database identifies the most common 3,000 words used in everyday English texts. If we use OEC, the vector length for each word could be $1 \times 3,000$. If we want to mimic a well-educated English speaker and we know that a native English speaker with a good education may use between 20,000 and 35,000 words(including less commonly used words, idiomatic expressions, and specialized vocabulary) the vector length could be as large as $1 \times 35,000$ for each word.

$$This = [0 \quad 0 \quad 1 \quad 0 \quad ...]$$

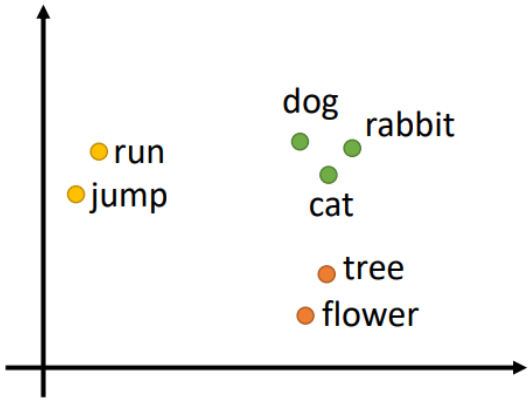
$$is = [1 \quad 0 \quad 0 \quad 0 \quad ...]$$

$$a = [0 \quad 1 \quad 0 \quad 0 \quad ...]$$

$$cat = [0 \quad 0 \quad 0 \quad 1 \quad ...]$$

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 by Word Embedding. Related words are transformed into vectors that have high correlations. Unrelated words are transformed into vectors that have low correlations. 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.



In particular,

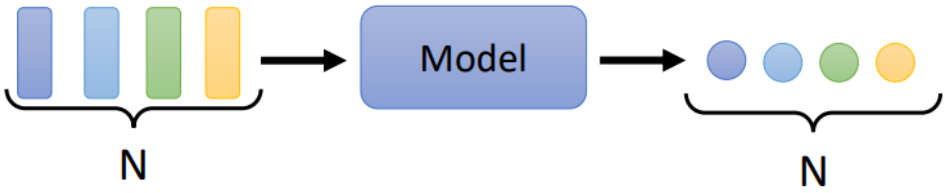
$$dog = [0.3645 \quad 0.9507 \quad 0.7320 \quad 0.5987]$$

$$cat = [0.2949 \quad 0.7559 \quad 0.6172 \quad 0.4943]$$

The correlation between the two vectors is approximately 0.996.

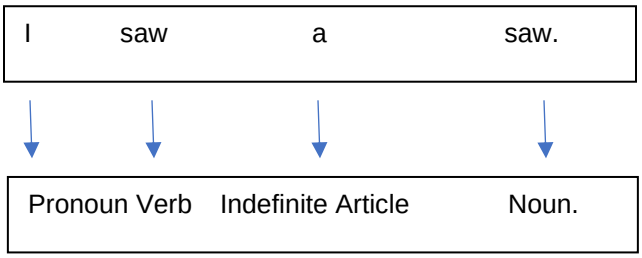
Output Types

Type 1 The output size is the same as the input. If the input is a sentence with N words (size of N), the output is a sequence of N outputs (size of N) too. In a POS tagging task, we need to determine the part of speech for each word, the model doesn't determine the output size.

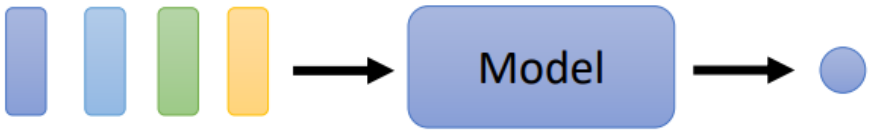


Example: POS Tagging

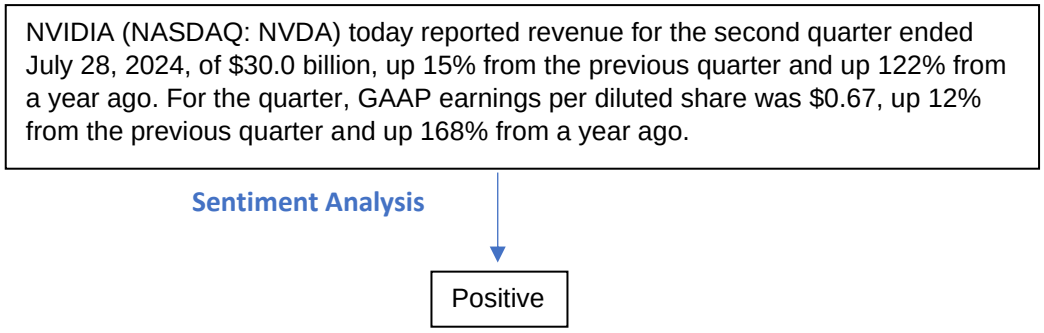
In corpus linguistics, part-of-speech tagging (POS tagging or PoS tagging or POST), also called grammatical tagging is the process of marking up a word in a text (corpus) as corresponding to a particular part of speech, based on both its definition and its context. The model needs to determine the part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



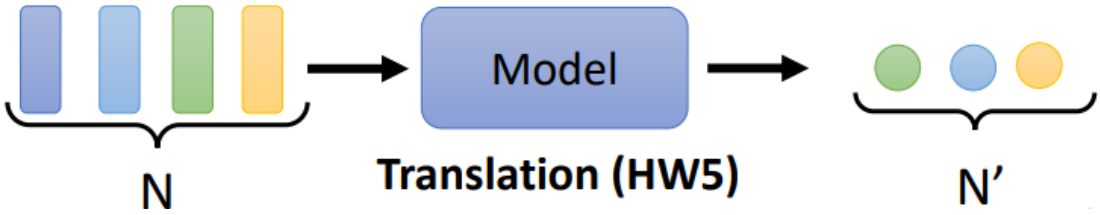
Type 2: There is one label for the entire output sequence.



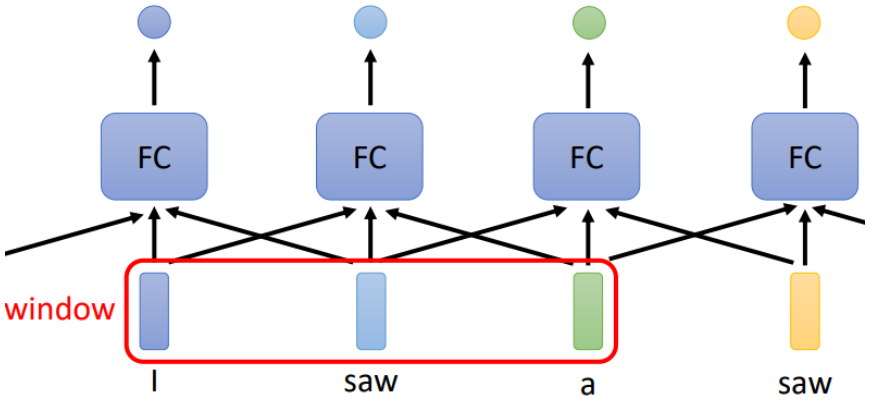
Example: Sentiment Analysis



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



Let's illustrate how self-attention works.

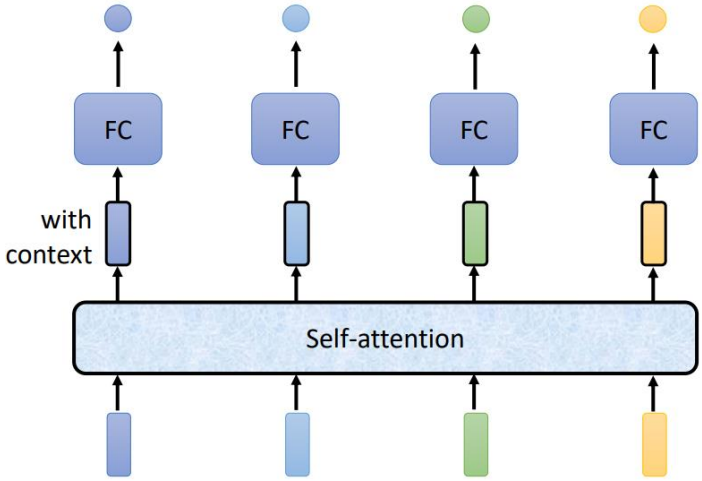


Solution 1: we add a window to connect the adjacent words.

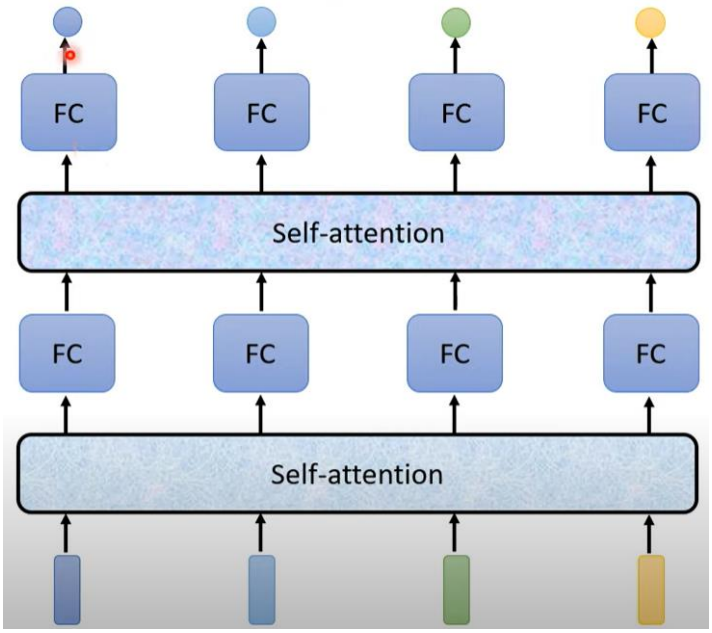
Solution 2: we connect the words in the entire input sequence using a large window.

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.

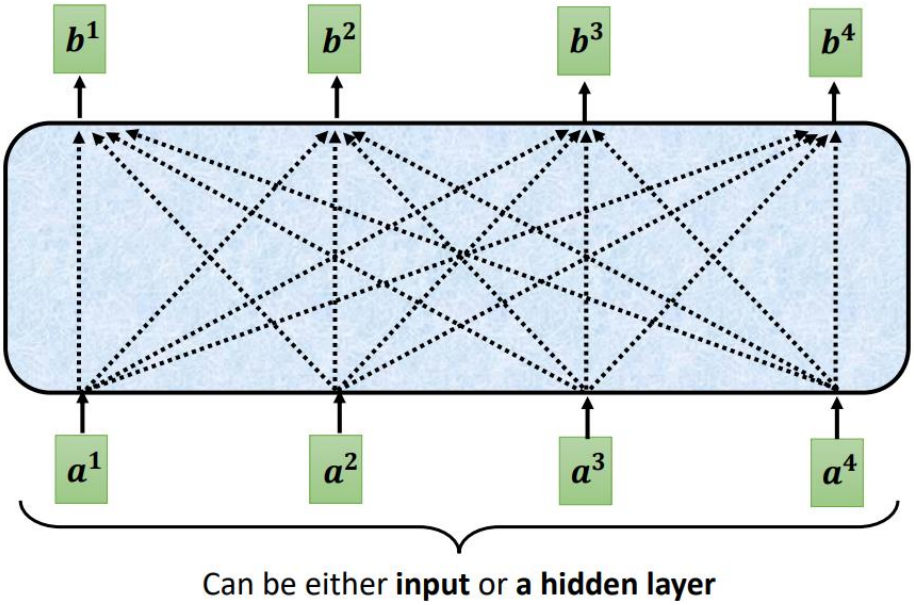


In fact, you could iteratively use the self-attention layer and FC layer several times to fully incorporate the context of the input sequence. FC focuses on processing the local information on a particular part of the input sequence. Self-attention aims to deal with the context of the entire input sequence. 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 Network

The self-attention can take either the raw input or the output from a hidden layer as input. In the diagram below, the input sequence is a sentence of four words, e.g. This is a cat.



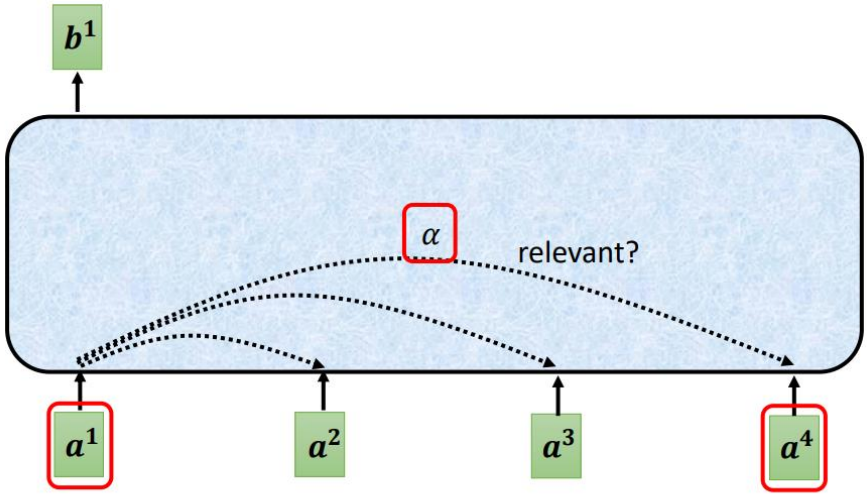
a^1, a^2, a^3, a^4 are input vectors fed into self-attention. Each represents a word from the sentence (e.g. a^4 represents "cat") and is the embedding vector of size $1 \times d$. We use a^i but not x^i to represent the input for self-attention because a^i may be the intermediate output from a particular 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 information from 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.

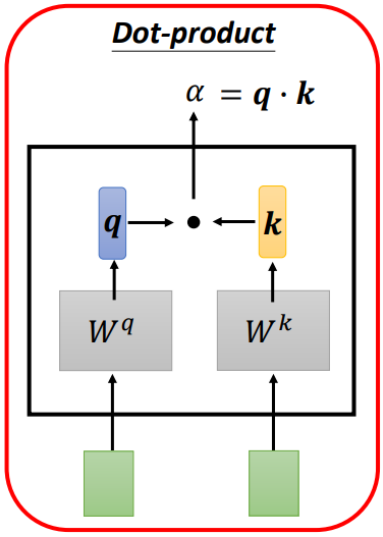
The purpose of self-attention is to find relevant vectors in a sequence. Given a^1, which among a^2, a^3, a^4 are relevant to determine the context of a^1. In the following example, we use a scalar $\alpha_{1,2}$ to represent the relevance (not correlation) between a^1 and a^2. The scalar α going through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



Find the relevant vectors in a sequence

Method 1: Dot Product

Here, the input a^1 is a $1 \times d$ vector. d is the hyperparameter d_{model}. In self-attention, the entire embedding vector is processed all at once. We use single-column vectors for query, key and value parameters. The query parameter matrix W^q is $d \times 1$. The key parameter matrix W^k is $d \times 1$. The relevance scalar α between the first word and itself is:



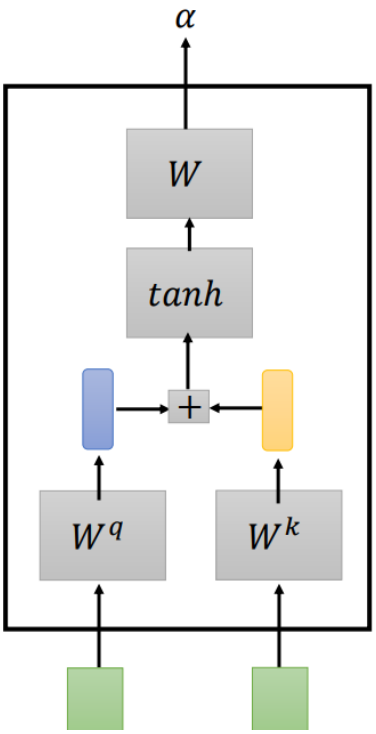
$$q^1 = a^1 W^q$$

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

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

- q represents query.
- k represents the key.
- Superscript and subscript represent the first word.

Method 2: Additive



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

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

$$\tilde{\alpha}_{1,1} = \tanh\left(q^1 + k^1\right)$$

$$a_{1,1} = W \cdot \tilde{\alpha}_{1,1}$$

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

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

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

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

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

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

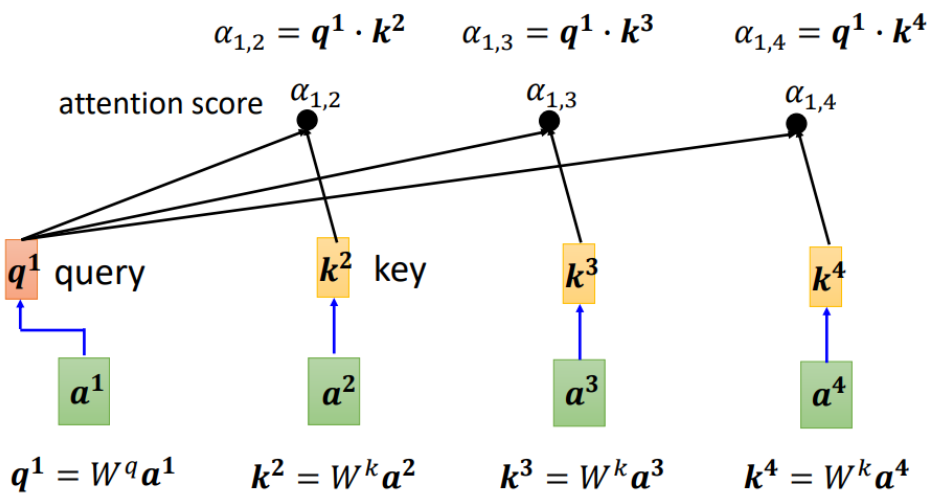
Here W^k and W^q are the same set of parameters used on a^1. The relevance score between a^1 and a^2, a^3, a^4 are:

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

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

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

The calculation process can be illustrated as follows.



In addition, we've calculated the attention score of a^1 with itself.

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

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

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

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

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

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

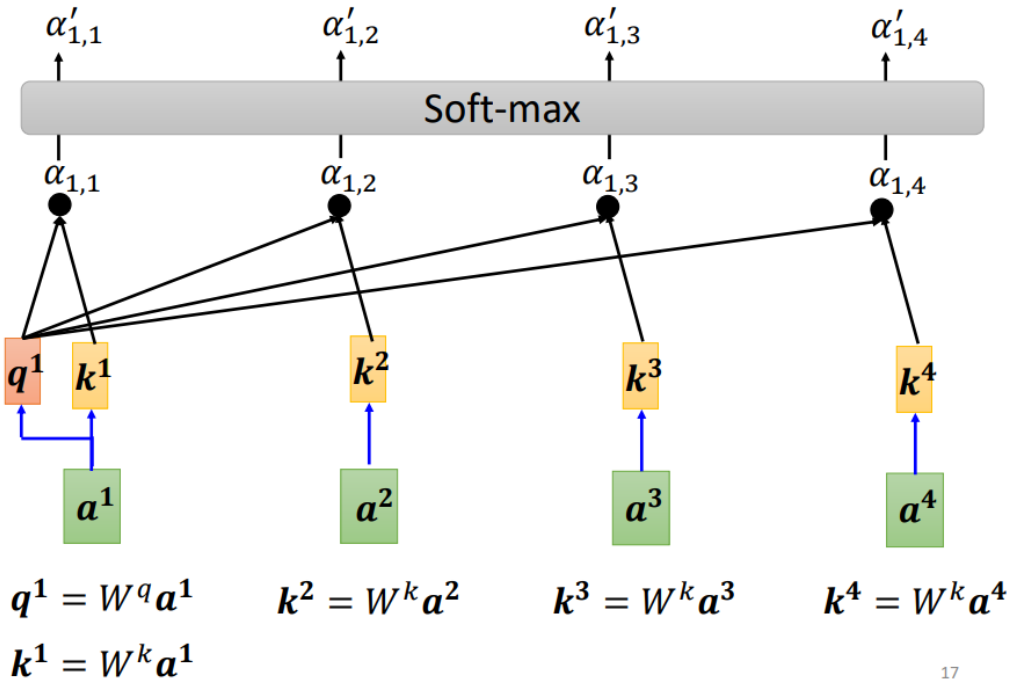
$$\alpha'_{1,3} = softmax\left(\alpha_{1,3}\right) = \frac{e^{\alpha_{1,3}}}{e^{\alpha_{1,1}} + e^{\alpha_{1,2}} + e^{\alpha_{1,3}} + e^{\alpha_{1,4}}}$$

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

In all, we could summarize the softmax process as:

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

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. We thus introduce another set of parameters W^v, which is also $d \times 1$.

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

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

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

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

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

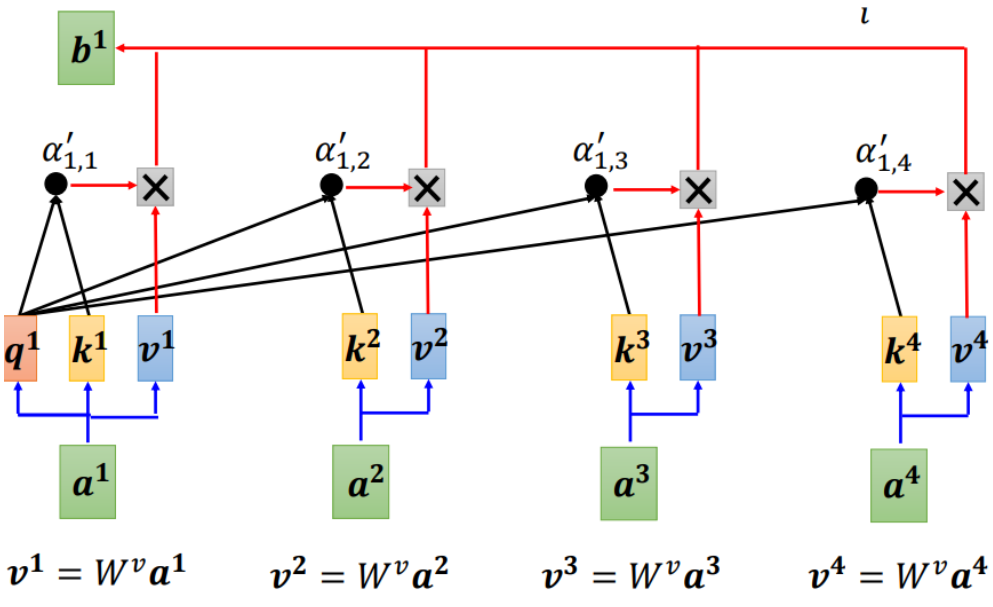
$$b^1 = \alpha'_{1,1} v^1 + \alpha'_{1,2} v^2 + \alpha'_{1,3} v^3 + \alpha'_{1,4} v^4$$

we could summarize the calculation as:

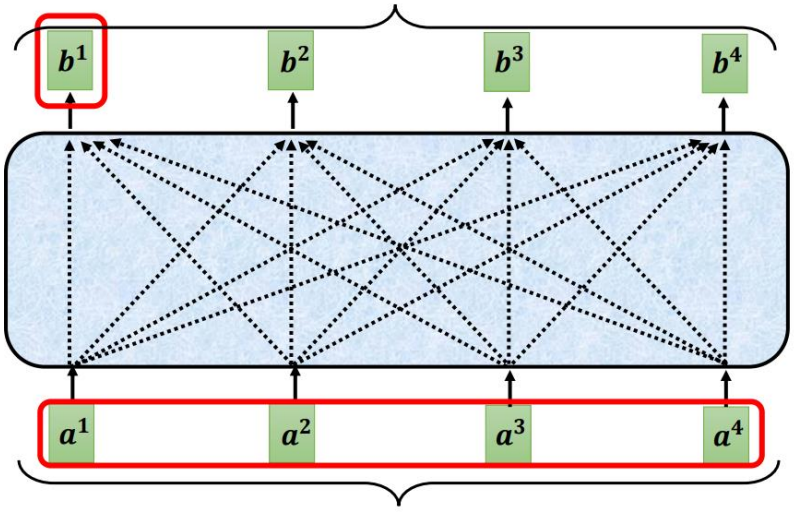
$$b^1 = \sum_{i=1}^{K} \alpha'_{1,i} \cdot v^i = [\alpha'_{1,1} \quad \alpha'_{1,2} \quad \alpha'_{1,3} \quad \alpha'_{1,4}] \begin{bmatrix} v^1 \\ v^2 \\ v^3 \\ v^4 \end{bmatrix} = \alpha'^T_1 v$$

Here the attention score works like a scalar. If an input a^i is highly relevant to a^1 , it should receive a high attention score $\alpha'_{1,i}$. The high attention score will help the meaning of the word v^i dominate the output b^1.

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.



Can be either **input** or **a hidden layer**

Let's repeat the above process to calculate b^2. In practice, you could parallel the calculation of b^i for $i = 1 \ldots N$ (all word in one sequence). First, calculate the query and the key for the second word

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

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

Next, calculate the attention between the second word and all other words in the sentence.

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

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

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

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

It is noteworthy that N sometimes does not refers to the full length of the sequence, but only defines the range the self-attention mechanism is attached to. For example, 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:

$$b^2 = \sum_{i=1}^{K} \alpha'_{2,i} \cdot v^i == \begin{bmatrix}\alpha'_{2,1} & \alpha'_{2,2} & \alpha'_{2,3} & \alpha'_{2,4}\end{bmatrix} \begin{bmatrix}v^1 \\ v^2 \\ v^3 \\ v^4\end{bmatrix} = \alpha'^T_2 v$$

Up to here, you may realize that given input a^1, a^2, a^3 and a^4, parameters of the network are:

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

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

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

Output of the network are $b: N \times 1$, a vector of attention scores.

$$b = A'\, v$$

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

In the $N \times N$ matrix A', $\alpha'_{i,j}$ is the softmax value of $\alpha_{i,j}$ (The softmax process is taken along each row). $\alpha_{i,j}$ is a 1×1 vector calculated by taking the dot product of query and key.

$$\alpha_{i,j} = q_i \cdot k_j$$

$q_i = a_i W^q$ and $k_j = a_j W^k$. The attention score matrix before the softmax is $A: N \times N$

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

It is noteworthy that the output of the self-attention network is not $b: N \times 1$. The output may take a different length N' than the input. We may introduce another set of parameters $W^O: N' \times N \times d'$ to obtain the final output of the self-attention network, $b': N' \times 1 \times d'$

$$b' = W^O b$$

You may now regard each row in b' is a vector of the probabilities. e.g. The vector size is $d' \times 1$ for each word in French/Italian, obtained from output (target) embedding. The probability will pin down the corresponding French/Italian word of translation.

5 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.

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 the query scalar q, the key scalar k and the meaning scalar v. Given the fact that each input a^i is a $1 \times d$ vector, attention mechanism processes the entire input sequence (all words simultaneously) into one $N \times d$ matrix. As we are processing a sentence with four words, $N = 4$.

$$\begin{bmatrix} q^1 \\ q^2 \\ q^3 \\ q^4 \end{bmatrix} = \begin{bmatrix} a_1^1 & \cdots & a_d^1 \\ \vdots & \ddots & \vdots \\ a_1^4 & \cdots & a_d^4 \end{bmatrix} W^q$$

$$\begin{bmatrix} k^1 \\ k^2 \\ k^3 \\ k^4 \end{bmatrix} = \begin{bmatrix} a_1^1 & \cdots & a_d^1 \\ \vdots & \ddots & \vdots \\ a_1^4 & \cdots & a_d^4 \end{bmatrix} W^k$$

$$\begin{bmatrix} v^1 \\ v^2 \\ v^3 \\ v^4 \end{bmatrix} = \begin{bmatrix} a_1^1 & \cdots & a_d^1 \\ \vdots & \ddots & \vdots \\ a_1^4 & \cdots & a_d^4 \end{bmatrix} W^v$$

$q^i: 1 \times M$, $k^i: 1 \times M$ and $v^i: 1 \times M$ for $i = 1, 2, 3, 4$. Parameters of the network are:

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

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

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

Calculate the inner product of q and k to obtain attention score. $K: N \times M$, $Q: N \times M$

$$A_1 = \begin{bmatrix} \alpha_{1,1} \\ \alpha_{1,2} \\ \alpha_{1,3} \\ \alpha_{1,4} \end{bmatrix} = \begin{bmatrix} k^1 \\ k^2 \\ k^3 \\ k^4 \end{bmatrix} \cdot q^1$$

$k^1: 1 \times M$ and $q^1: 1 \times M$. $\alpha_{1,1}: 1 \times M$. Attention score for the first word A_1 is an $N \times M$ matrix. Each row represents how the first word is related to the word in the row. Correspondingly, Attention score for the second word A_2 is an $N \times M$ matrix. Each row represents how the second word is related to the word in the row.

$$A_2 = \begin{bmatrix} \alpha_{2,1} \\ \alpha_{2,2} \\ \alpha_{2,3} \\ \alpha_{2,4} \end{bmatrix} = \begin{bmatrix} k^1 \\ k^2 \\ k^3 \\ k^4 \end{bmatrix} \cdot q^2$$

which is to say,

$$A = < K, Q > = \begin{bmatrix} \alpha_{1,1} & \alpha_{2,1} & \alpha_{3,1} & \alpha_{4,1} \\ \alpha_{1,2} & \alpha_{2,2} & \alpha_{3,3} & \alpha_{4,2} \\ \alpha_{1,3} & \alpha_{2,3} & \alpha_{3,3} & \alpha_{4,3} \\ \alpha_{1,4} & \alpha_{2,4} & \alpha_{3,4} & \alpha_{4,4} \end{bmatrix}$$

Each element is a $1 \times M$ vector.

Taking softmax, the sum of each column must be 1. Each element is still a $1 \times M$ vector.

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

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

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

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

$v^1: 1 \times M$ and $\alpha'_{1,1}: 1 \times M$. $b^1: N \times NM$

i.e.

$$O =< V, A >$$

To summarize the above procedure,

$$Q = X\,W^q$$

$$K = X\,W^k$$

$$V = X\,W^v$$

Next:

$$A =< K, Q >$$

The attention matrix is:

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

The output is:

$$O =< V, A >$$

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

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:

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

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

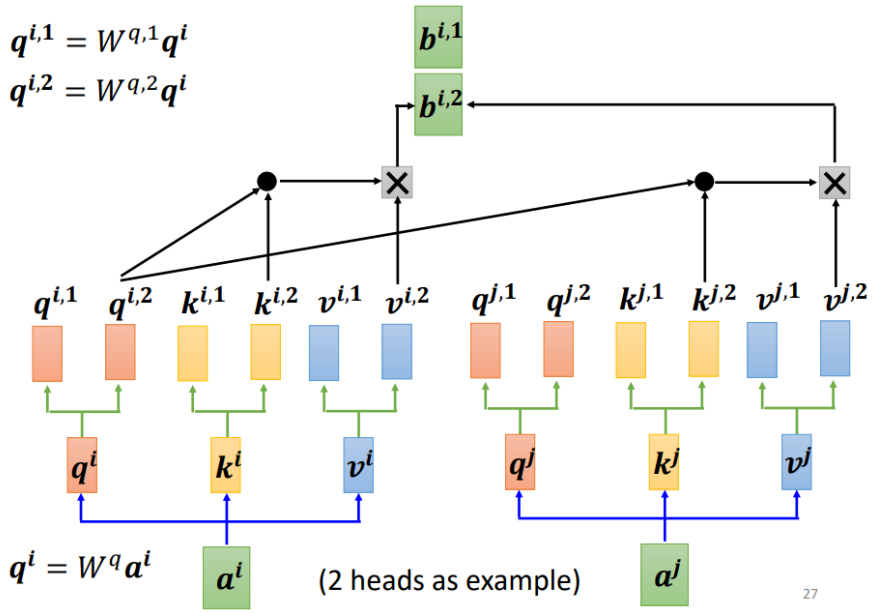
$$\alpha'_{i,i,1} = softmax(\alpha_{i,i,1})$$

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

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

...

The above procedure could be illustrated as follows.



Finally

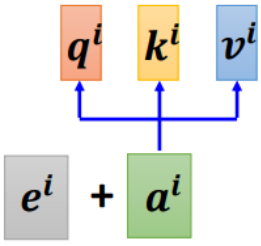
$$b^i = W^O \cdot \begin{bmatrix} b^{i,1} \\ b^{i,2} \end{bmatrix}$$

The output to calculate the loss function must be the same size as the target (and on most occasions the input too - at least the length of the embedding vector must be the same).

6 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.



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

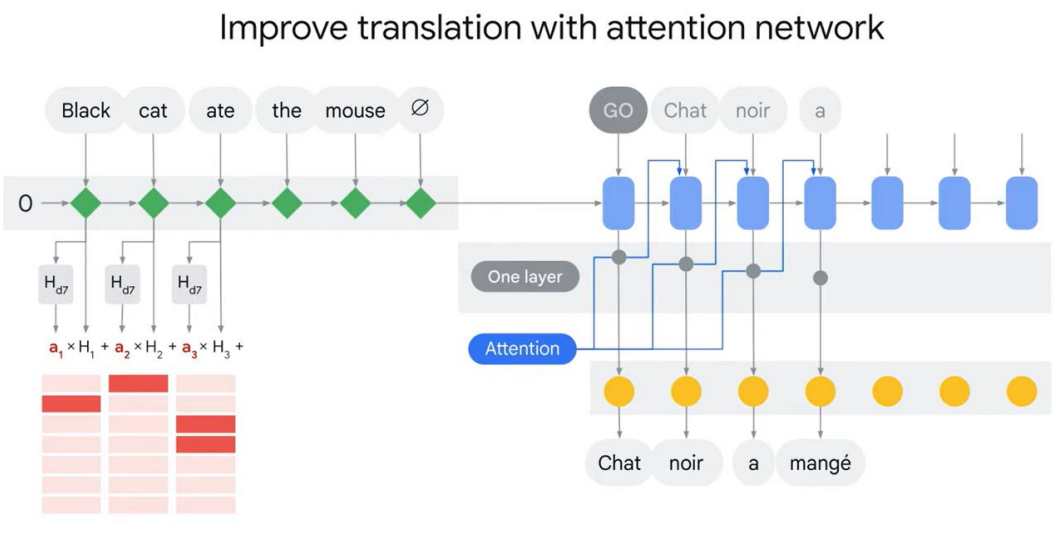
- In the context of autoencoders, the encoder is a neural network that takes an input (such as an image, text, or some other data representation) and compresses it into a latent space representation, often of lower dimensionality than the input.
- In sequence-to-sequence models like those used in machine 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.

A2 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.

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