Intro to Neural Nets

Attention & Transformers

Today's Agenda

Attention Mechanism in RNNs for Translation (2014)

- Sequence to sequence models with attention in translation tasks.

More General, Approach (No RNNs Required)

- We can hard-code a measure of semantic similarity between a focal token, and all other tokens in the sequence

QKV Framework & Multi-head Attention

- We can incorporate trainable weights around the attention mechanism.

TF-IDF is “Static Attention”

Not all phrases are of equal importance…

- E.g., David less important than Beckham
- If a term occurs all the time, observing its presence is less informative

Inverse-document frequency (IDF) helps address this.

$$\mathrm{IDF} = \log(N/n_j)$$

- Term ‘weighting’ is then calculated as Term Frequency (TF) x IDF
- n_j= # of docs containing the term, N = total # of docs
- A term is deemed important if it has a high TF and/or a high IDF.
- As TF goes up, the word is more common generally. As IDF goes up, it means very few documents contain this term.

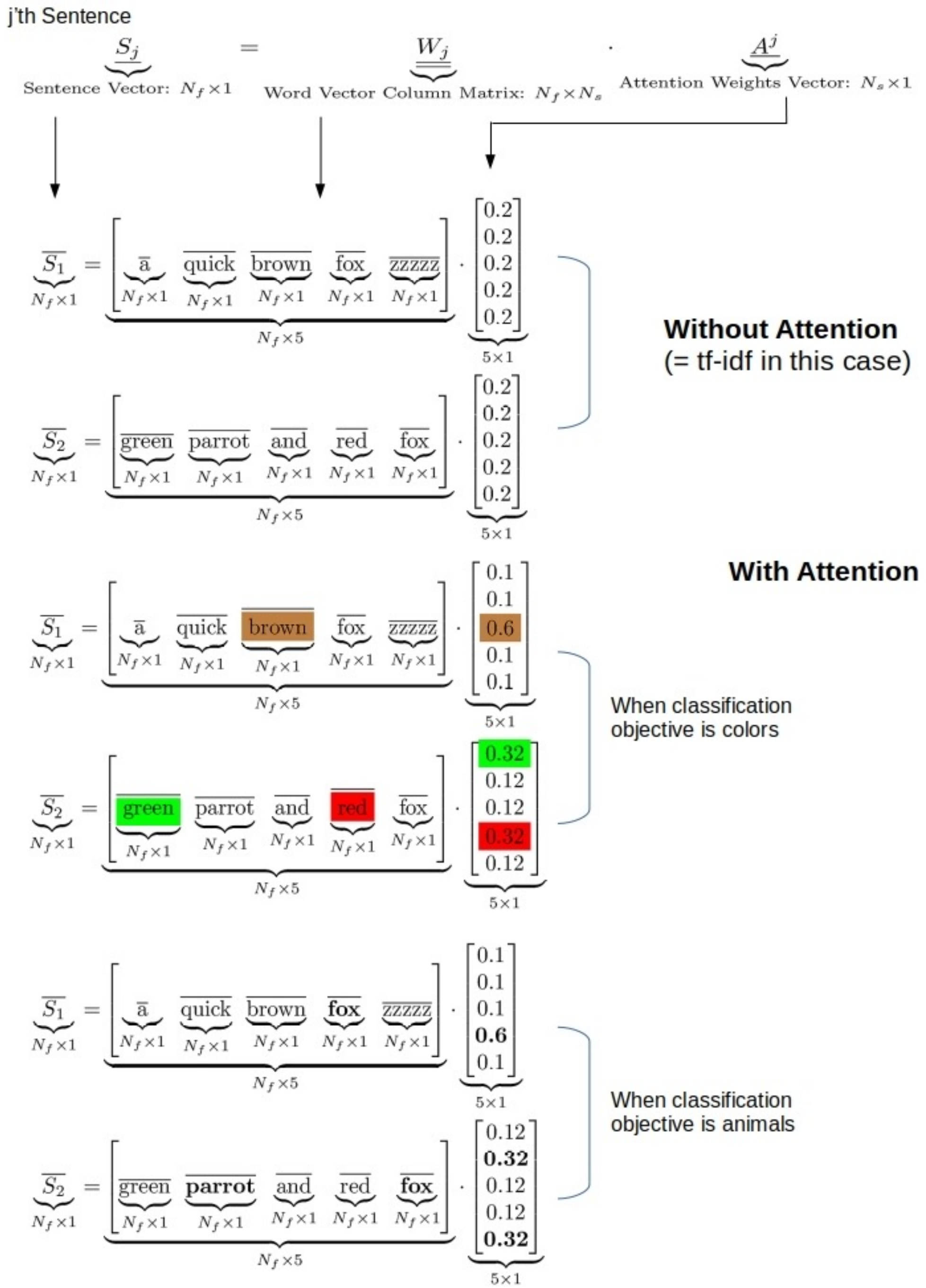
What Would Adaptive Weights Look Like?

Figure Out Weights Useful for Prediction

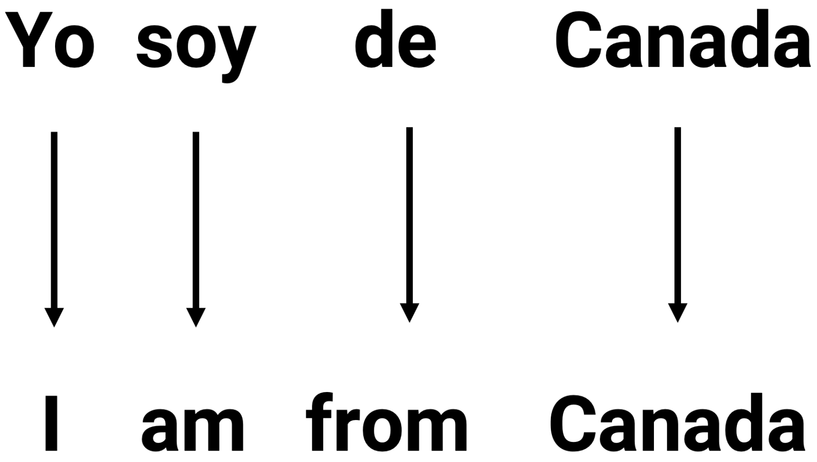
- Rather than telling the network the weights, which is what TF-IDF effectively does, we provide the network a weight matrix to update!

This Yields Task-specific Token Weights

- See some simple examples on the right.
- Our network may learn that certain tokens are more important for predicting a given label.

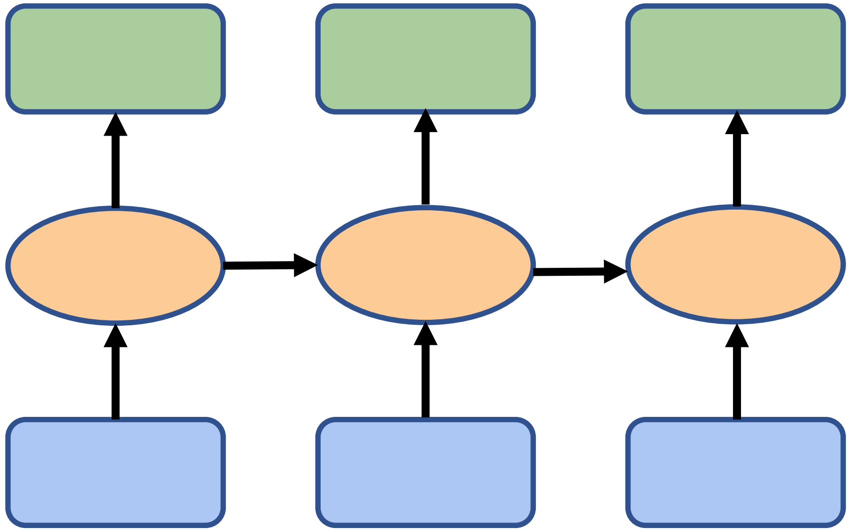

j'th Sentence
$S_j = W_j \cdot A^j$
Sentence Vector: $N_f \times 1$
Word Vector Column Matrix: $N_f \times N_s$
Attention Weights Vector: $N_s \times 1$
S_1 = a quick brown fox zzzzz · [0.2 0.2 0.2 0.2 0.2]
S_2 = green parrot and red fox · [0.2 0.2 0.2 0.2 0.2]
Without Attention
(= tf-idf in this case)
With Attention
S_1 = a quick brown fox zzzzz · [0.1 0.1 0.6 0.1 0.1]
S_2 = green parrot and red fox · [0.32 0.12 0.12 0.32 0.12]
When classification objective is colors
S_1 = a quick brown fox zzzzz · [0.1 0.1 0.1 0.6 0.1]
S_2 = green parrot and red fox · [0.12 0.32 0.12 0.12 0.32]
When classification objective is animals
$N_f \times 1$
$N_f \times 5$
5×1

Where Attention Came From:
Sequence-to-Sequence Models

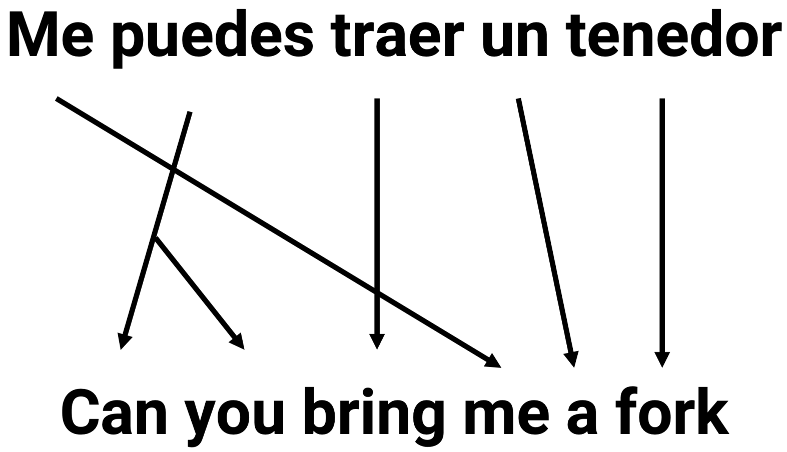


Yo soy de Canada

I am from Canada

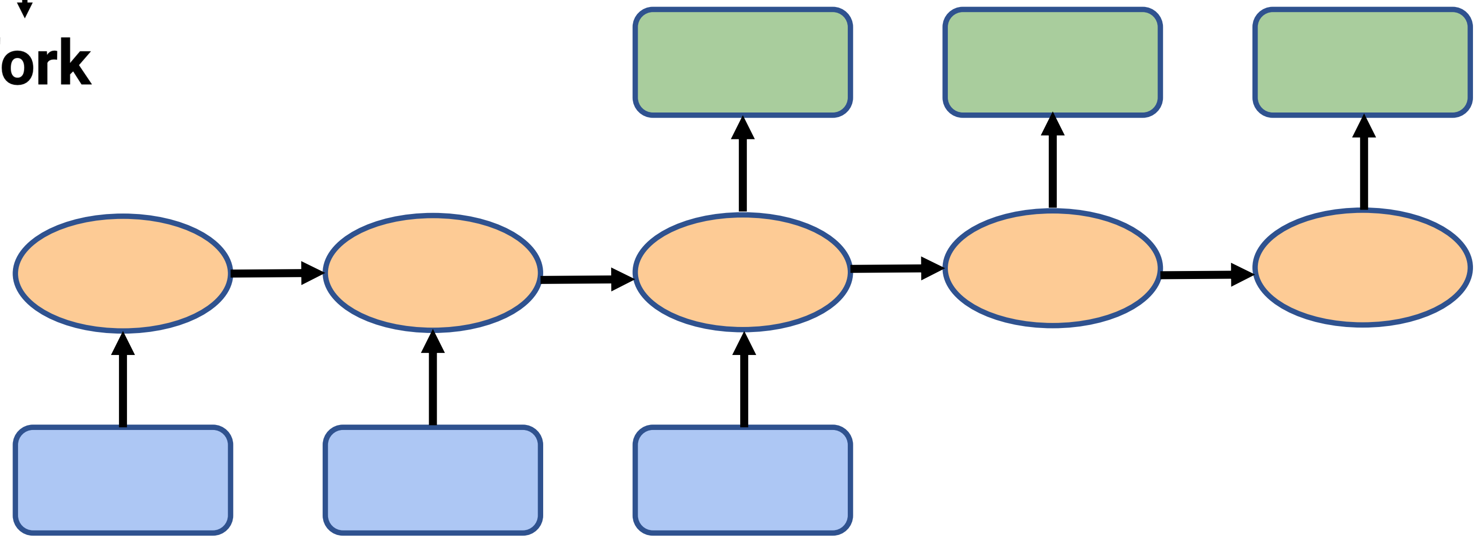


Struggling to Remember Long Sequences

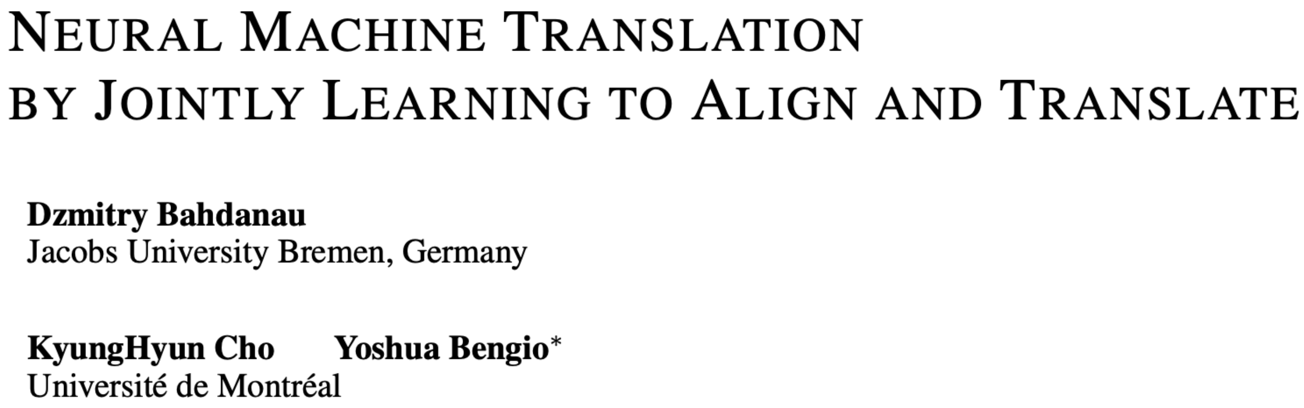


Me puedes traer un tenedor

Can you bring me a fork



The First (RNN) Attention Model



NEURAL MACHINE TRANSLATION
BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

Dzmitry Bahdanau
Jacobs University Bremen, Germany

KyungHyun Cho **Yoshua Bengio**[*]
Université de Montréal

" … allowing a model to automatically (soft-)search for parts of a source sentence that are relevant to predicting a target word …"

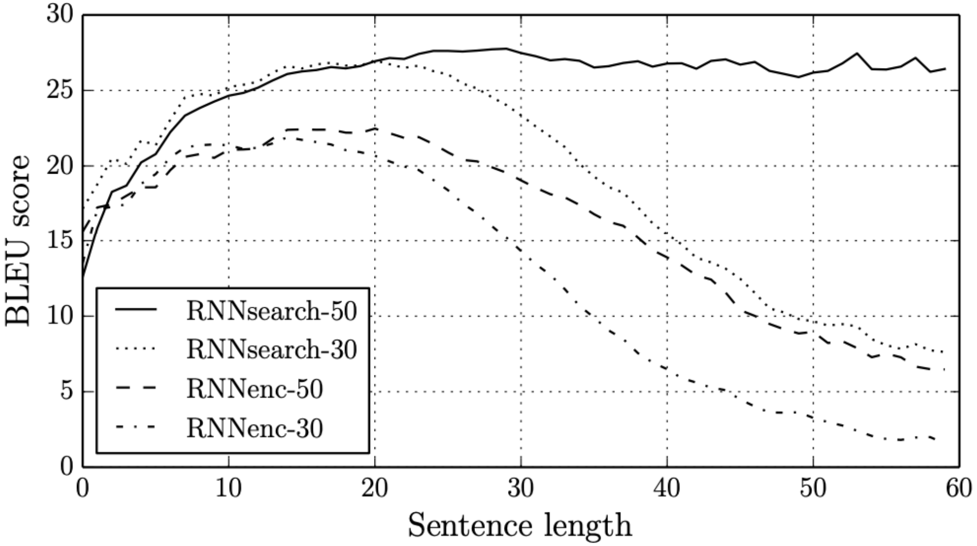


Figure 2: The BLEU scores of the generated translations on the test set with respect to the lengths of the sentences. The results are on the full test set which includes sentences having unknown words to the models.

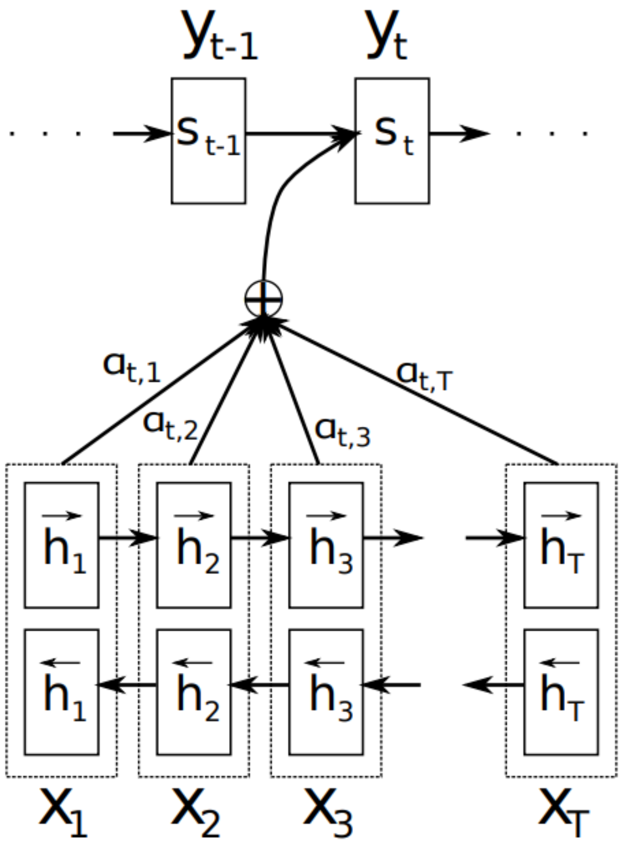
The First (RNN) Attention Model

Two RNNs

- A bidirectional RNN handling the input sentence (encoder).
- A regular RNN handling the output (decoder).

Two RNNs

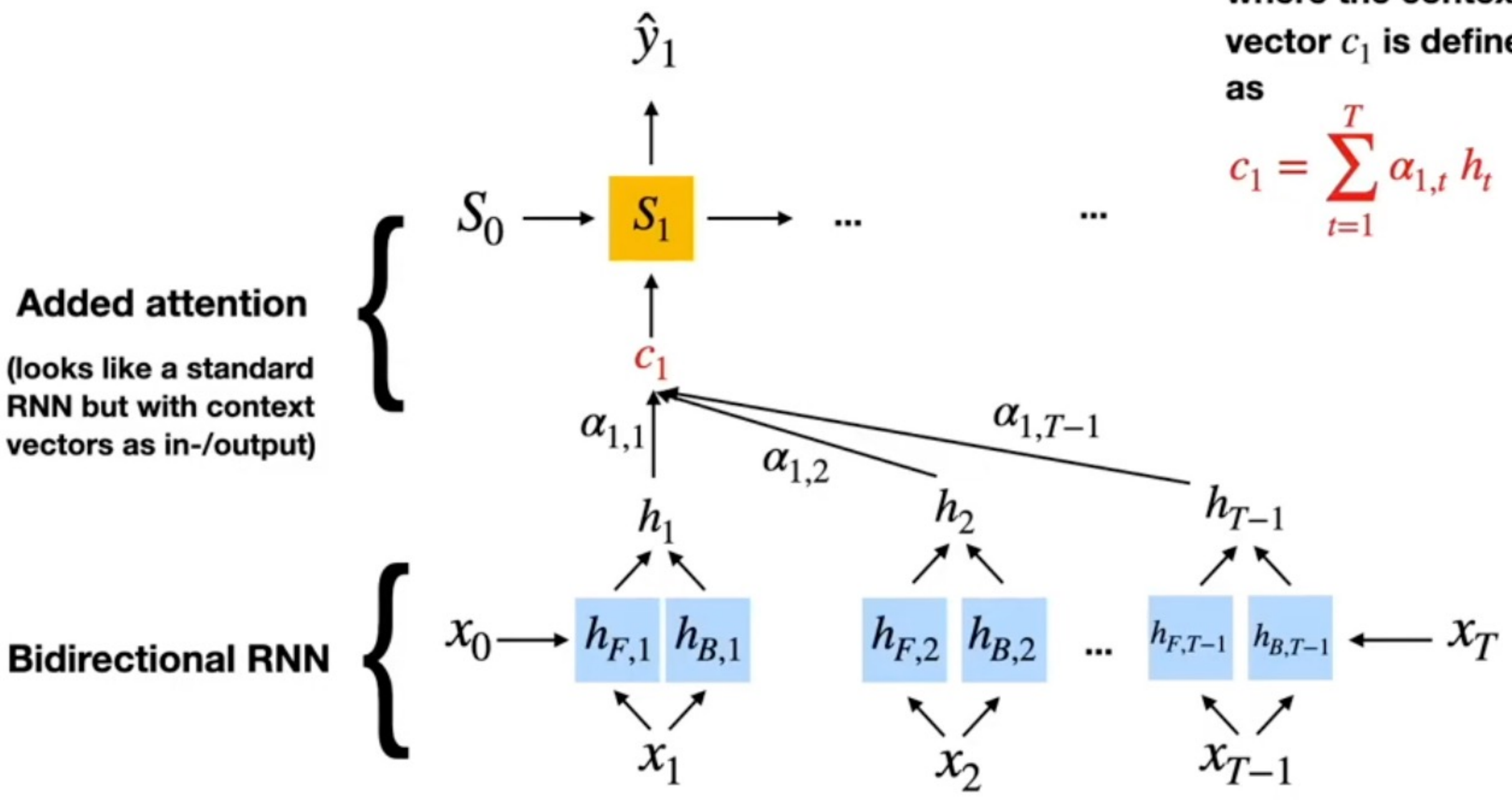
- Bidirectional RNN reads sentence in both directions.
- Produces a hidden state output for each token of input.
- Attention weights are calculated too, per token, based on the entire input sequence.
- Finally, the attention-weighted output for each token is passed into the regular RNN (which does the final translation).
- Each output from that last RNN is based on attention-weighted input token and output hidden state of last translated token (i.e., s_t-1)



y_{t-1}
y_t
s_{t-1}
s_t
$\alpha_{t,1}$
$\alpha_{t,2}$
$\alpha_{t,3}$
$\alpha_{t,T}$
$\overrightarrow{h}_1$ $\overrightarrow{h}_2$ $\overrightarrow{h}_3$ $\overrightarrow{h}_T$
$\overleftarrow{h}_1$ $\overleftarrow{h}_2$ $\overleftarrow{h}_3$ $\overleftarrow{h}_T$
x_1 x_2 x_3 x_T

Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.

The First (RNN) Attention Model



Added attention

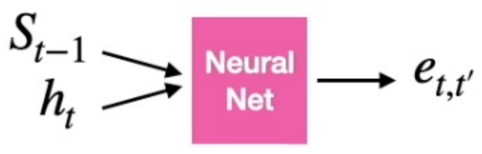
(looks like a standard RNN but with context vectors as in-/output)

Bidirectional RNN

where the context vector c_1 is defined as

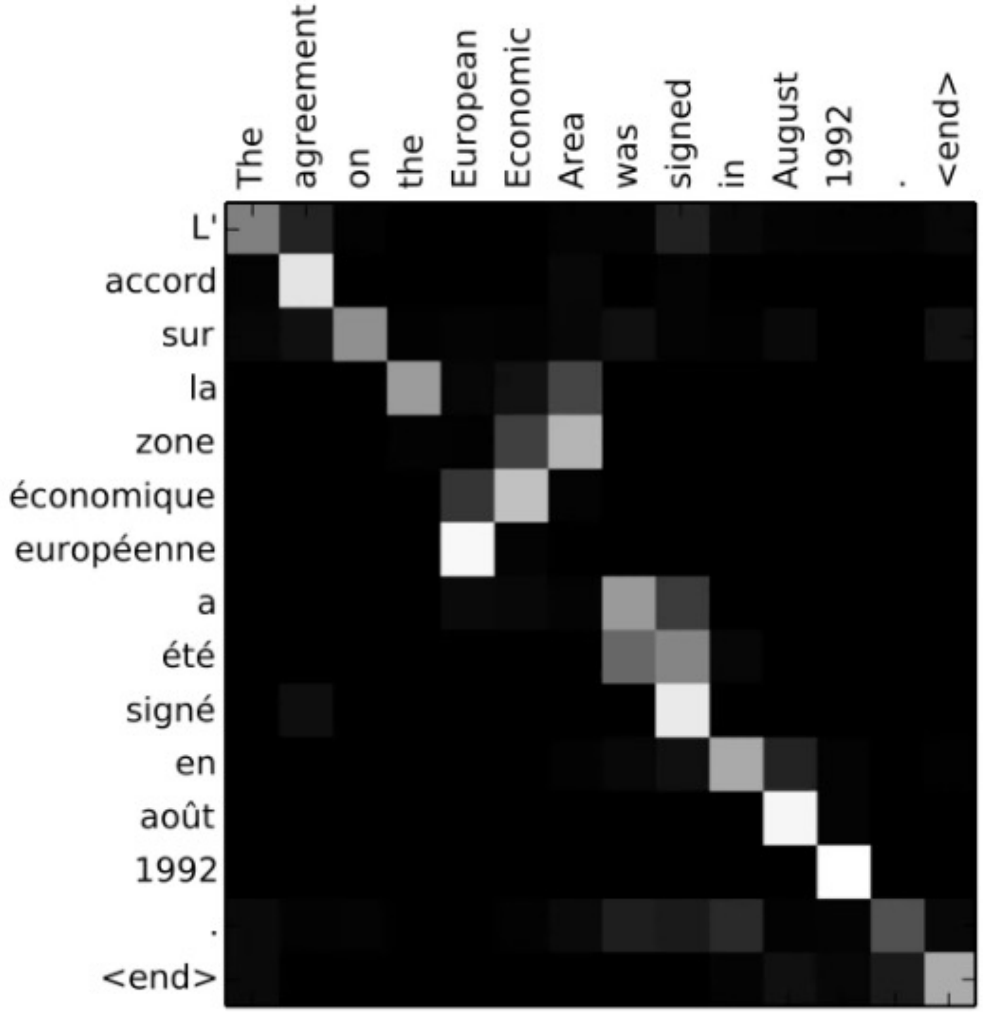
$$c_1 = \sum_{t=1}^{T} \alpha_{1,t}\, h_t$$

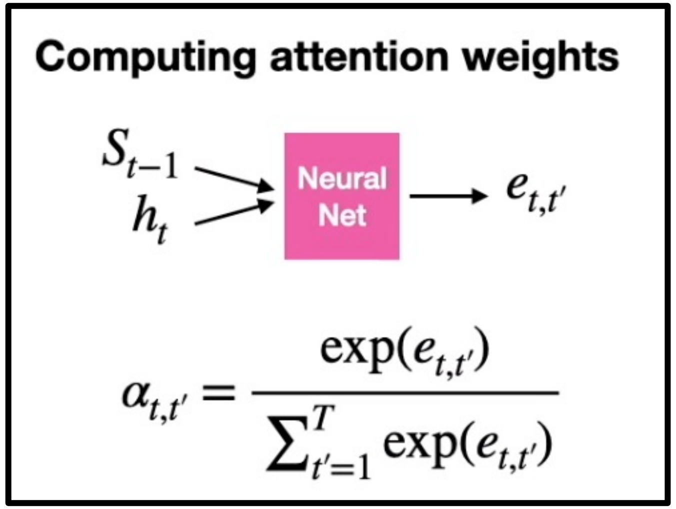
Computing attention weights



$$\alpha_{t,t'} = \frac{\exp(e_{t,t'})}{\sum_{t'=1}^{T} \exp(e_{t,t'})}$$

The First (RNN) Attention Model




Computing attention weights

$$\alpha_{t,t'} = \frac{\exp(e_{t,t'})}{\sum_{t'=1}^{T} \exp(e_{t,t'})}$$

We Don’t Need an RNN to Do This

We Can Drop the RNN and Process the Whole Sequence in Parallel

- Note that transformers work with a similar autoencoder architecture, but no LSTMs needed (we use ”stacked attention layers”).
- Transformers are actually more computationally expensive (more calculations required), but because we can parallelize them they can be trained faster than RNNs.
- By moving away from RNN specific implementation we can also think about more general forms of attention (e.g.,. For CNNs).

Attention Is All You Need

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

Jakob Uszkoreit*
Google Research
usz@google.com

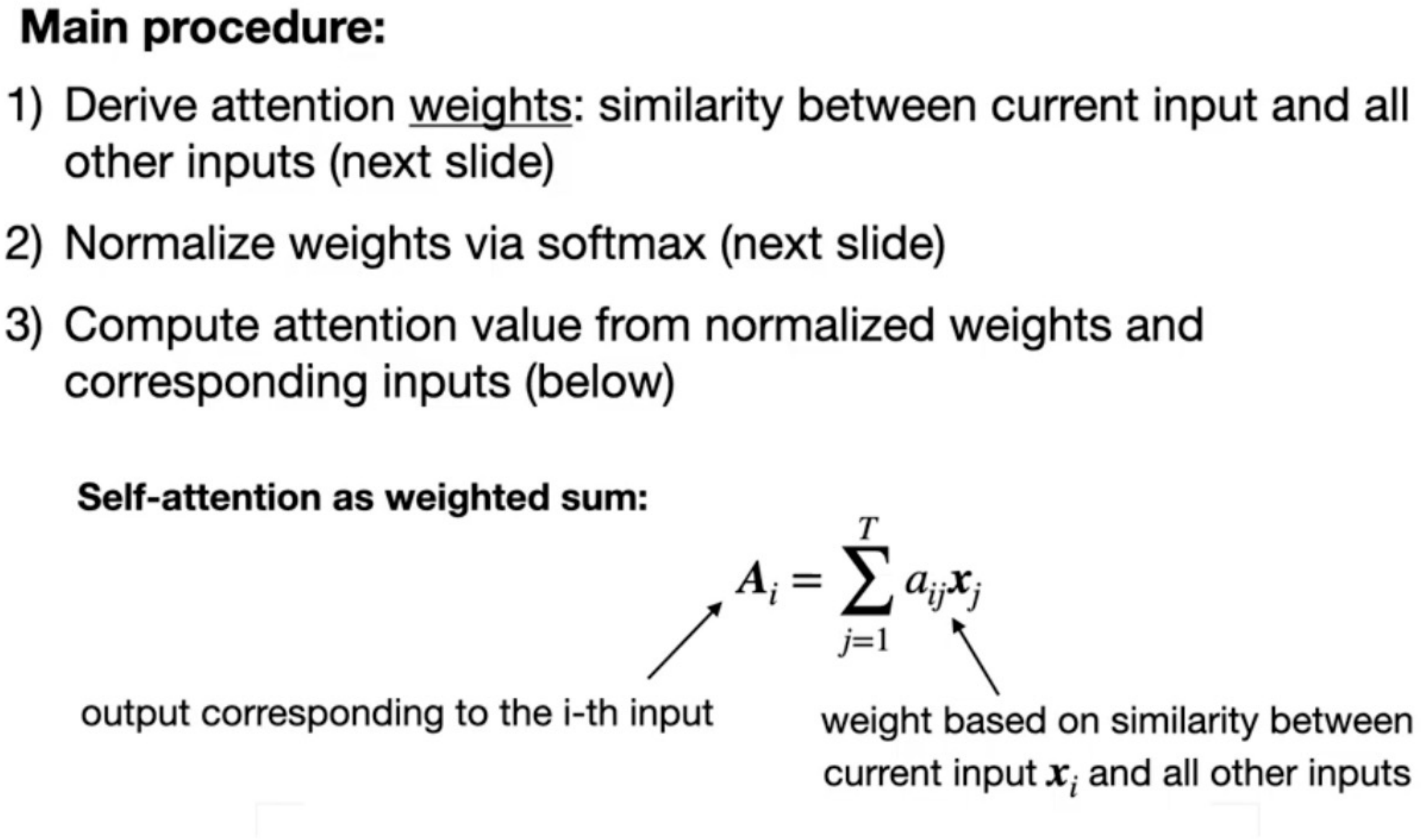
Llion Jones*
Google Research
llion@google.com

Aidan N. Gomez* †
University of Toronto
aidan@cs.toronto.edu

Łukasz Kaiser*
Google Brain
lukaszkaiser@google.com

Illia Polosukhin* ‡
illia.polosukhin@gmail.com

A Simpler Attention Mechanism



Main procedure:

1) Derive attention weights: similarity between current input and all other inputs (next slide)
2) Normalize weights via softmax (next slide)
3) Compute attention value from normalized weights and corresponding inputs (below)

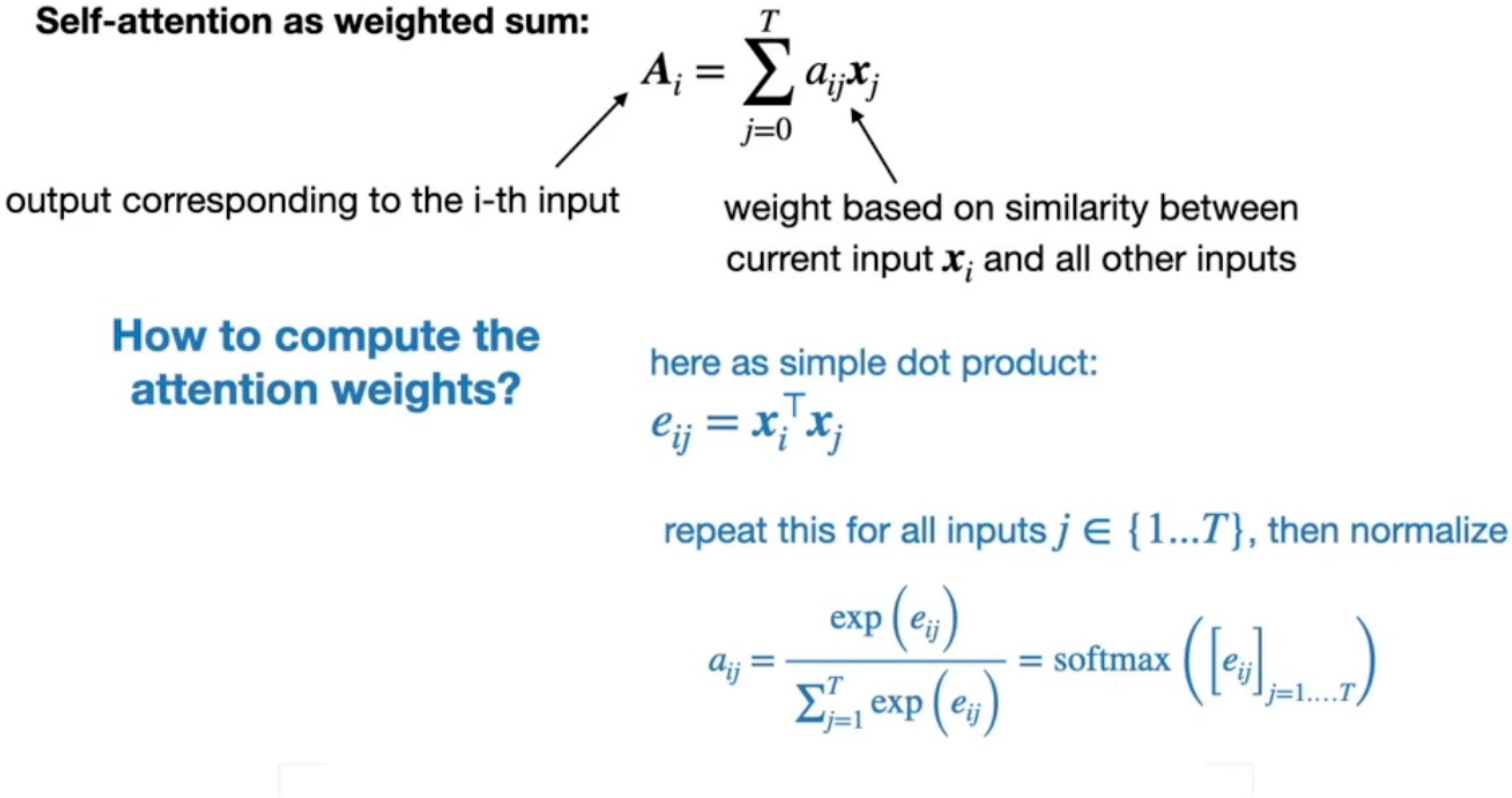
Self-attention as weighted sum:

$$\boldsymbol{A}_i = \sum_{j=1}^{T} a_{ij}\boldsymbol{x}_j$$

output corresponding to the i-th input

weight based on similarity between current input $\boldsymbol{x}_i$ and all other inputs

A Simple Attention Mechanism



Self-attention as weighted sum:

$$\boldsymbol{A}_i = \sum_{j=0}^{T} a_{ij} \boldsymbol{x}_j$$

output corresponding to the i-th input

weight based on similarity between current input $\boldsymbol{x}_i$ and all other inputs

How to compute the attention weights?

here as simple dot product:

$$e_{ij} = \boldsymbol{x}_i^\top \boldsymbol{x}_j$$

repeat this for all inputs $j \in \{1 \ldots T\}$, then normalize

$$a_{ij} = \frac{\exp\left(e_{ij}\right)}{\sum_{j=1}^{T} \exp\left(e_{ij}\right)} = \text{softmax}\left(\left[e_{ij}\right]_{j=1 \ldots T}\right)$$

A Simple Attention Mechanism

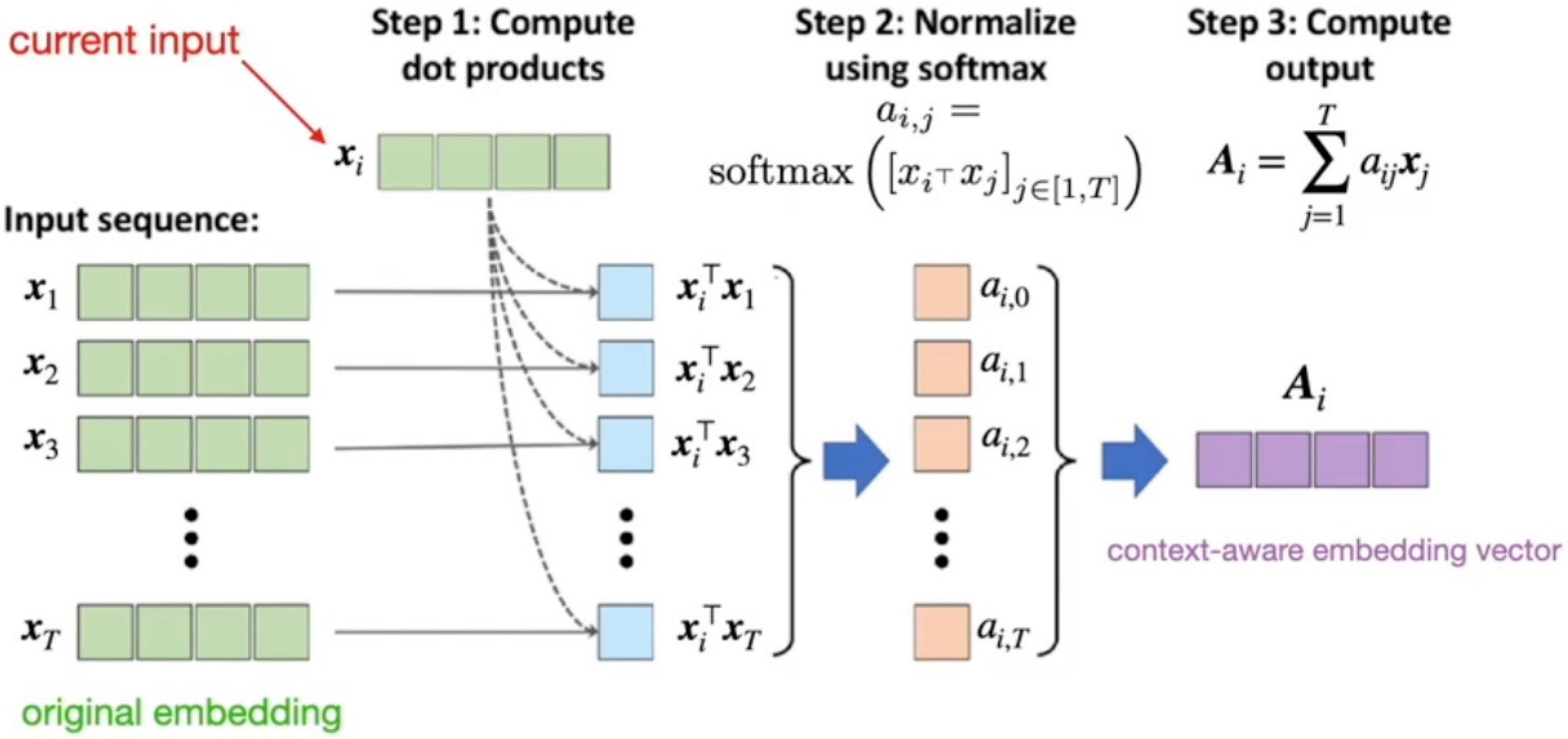
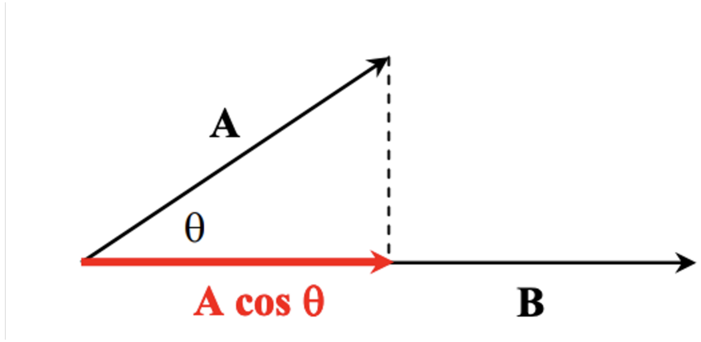

current input
Step 1: Compute dot products
Step 2: Normalize using softmax
$a_{i,j} = \text{softmax}\left([x_i^\top x_j]_{j\in[1,T]}\right)$
Step 3: Compute output
$\boldsymbol{A}_i = \sum_{j=1}^{T} a_{ij}\boldsymbol{x}_j$
$\boldsymbol{x}_i$
Input sequence:
$\boldsymbol{x}_1$
$\boldsymbol{x}_2$
$\boldsymbol{x}_3$
$\boldsymbol{x}_T$
$\boldsymbol{x}_i^\top \boldsymbol{x}_1$
$\boldsymbol{x}_i^\top \boldsymbol{x}_2$
$\boldsymbol{x}_i^\top \boldsymbol{x}_3$
$\boldsymbol{x}_i^\top \boldsymbol{x}_T$
$a_{i,0}$
$a_{i,1}$
$a_{i,2}$
$a_{i,T}$
$\boldsymbol{A}_i$
context-aware embedding vector
original embedding

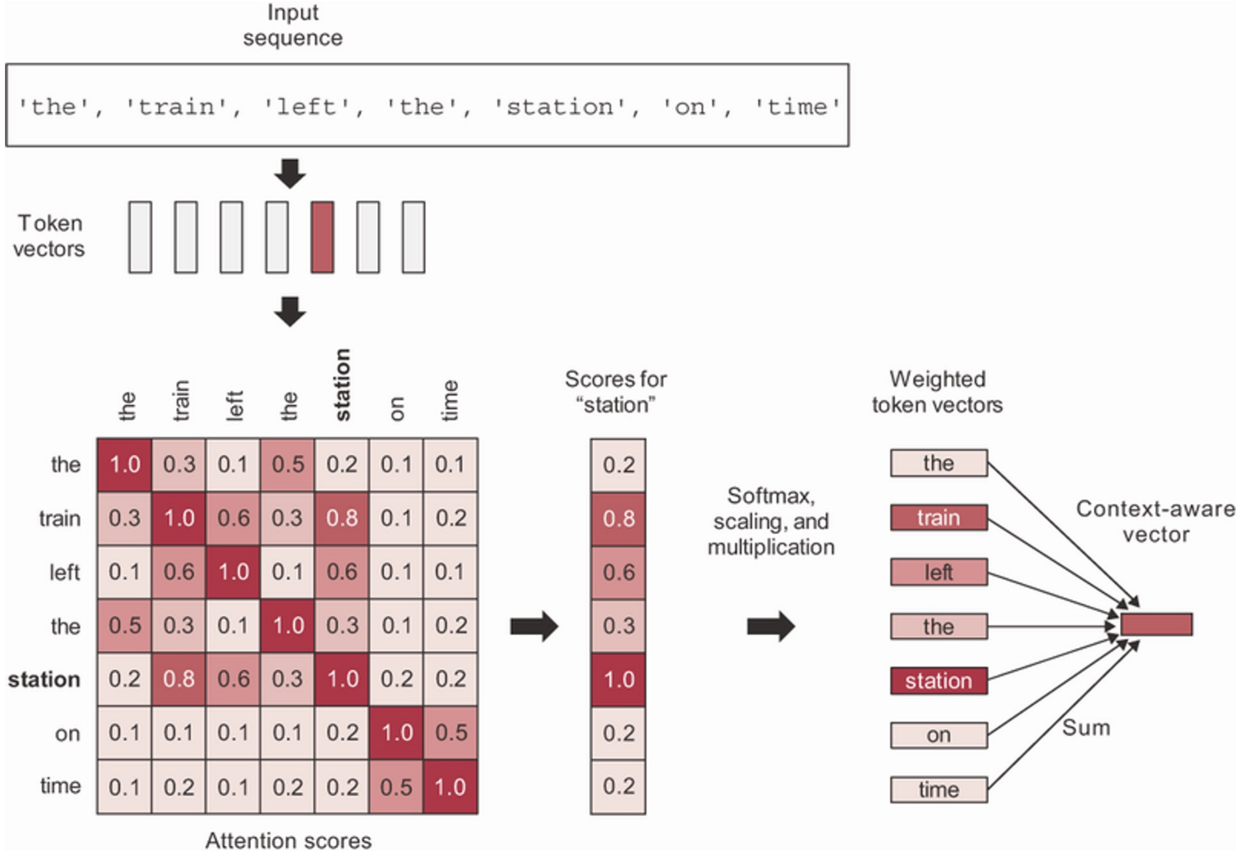
Image source: Raschka & Mirjalili 2019. Python Machine Learning, 3rd edition

A Basic Self-Attention Mechanism

Most Basic Form of Self-Attention

- This is what the textbook introduces.
- We magnify vector representations of tokens based on how parallel (semantically related) they are to the other tokens in the input sequence.





	the	train	left	the	**station**	on	time
the	1.0	0.3	0.1	0.5	0.2	0.1	0.1
train	0.3	1.0	0.6	0.3	0.8	0.1	0.2
left	0.1	0.6	1.0	0.1	0.6	0.1	0.1
the	0.5	0.3	0.1	1.0	0.3	0.1	0.2
station	0.2	0.8	0.6	0.3	1.0	0.2	0.2
on	0.1	0.1	0.1	0.1	0.2	1.0	0.5
time	0.1	0.2	0.1	0.2	0.2	0.5	1.0

Attention scores

Scores for “station”
0.2
0.8
0.6
0.3
1.0
0.2
0.2

The Basic Attention Mechanism is Rigid

The Approach Considers the Input Context in a Very ‘Fixed’ Manner

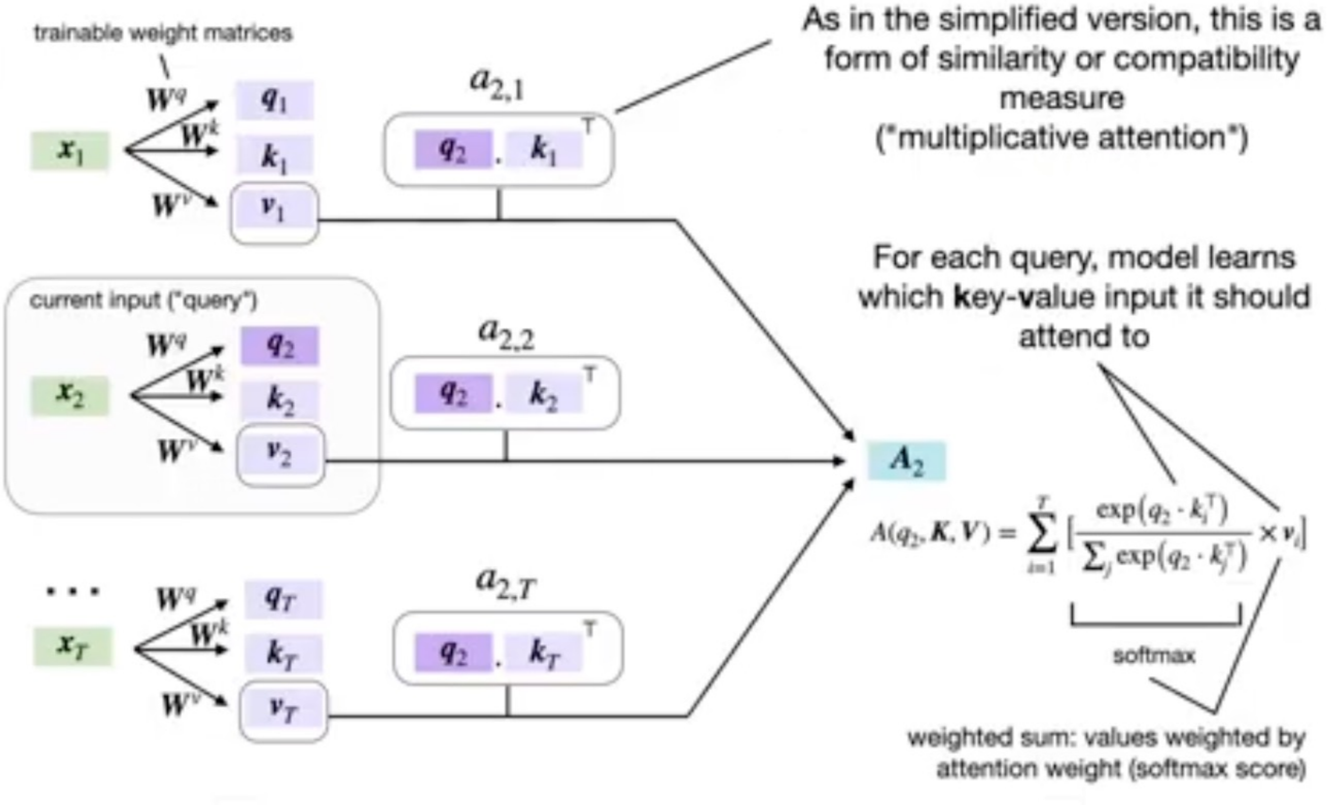
- We get context-specific attention weights, but the notion of context is pre-defined / hard-coded in the dot-product-summation procedure.
- The pre-defined approach to attention does not generalize outside of language.
- The pre-defined approach to attention assumes that shared semantic similarity is the best heuristic for attention, but perhaps it is not in some cases. Most obviously, perhaps in some scenarios it is ideal to focus attention on words that are semantically dissimilar.
- How can we enable a procedure of this sort, but using trainable parameters, so the network can figure out how to allocate attention by itself?

Scaled Dot-Product Attention Mechanism

query = $\boldsymbol{W}^q \boldsymbol{x}_i$

key = $\boldsymbol{W}^k \boldsymbol{x}_i$

value = $\boldsymbol{W}^v \boldsymbol{x}_i$



As in the simplified version, this is a form of similarity or compatibility measure ("multiplicative attention")

For each query, model learns which **k**ey-**v**alue input it should attend to

$$A(q_2, \boldsymbol{K}, \boldsymbol{V}) = \sum_{i=1}^{T} [\underbrace{\frac{\exp(q_2 \cdot k_i^\top)}{\sum_j \exp(q_2 \cdot k_j^\top)}}_{\text{softmax}} \times \boldsymbol{v}_i]$$

weighted sum: values weighted by attention weight (softmax score)

Flexible Contextual Attention Mechanism

Scaled Dot-Product Attention…

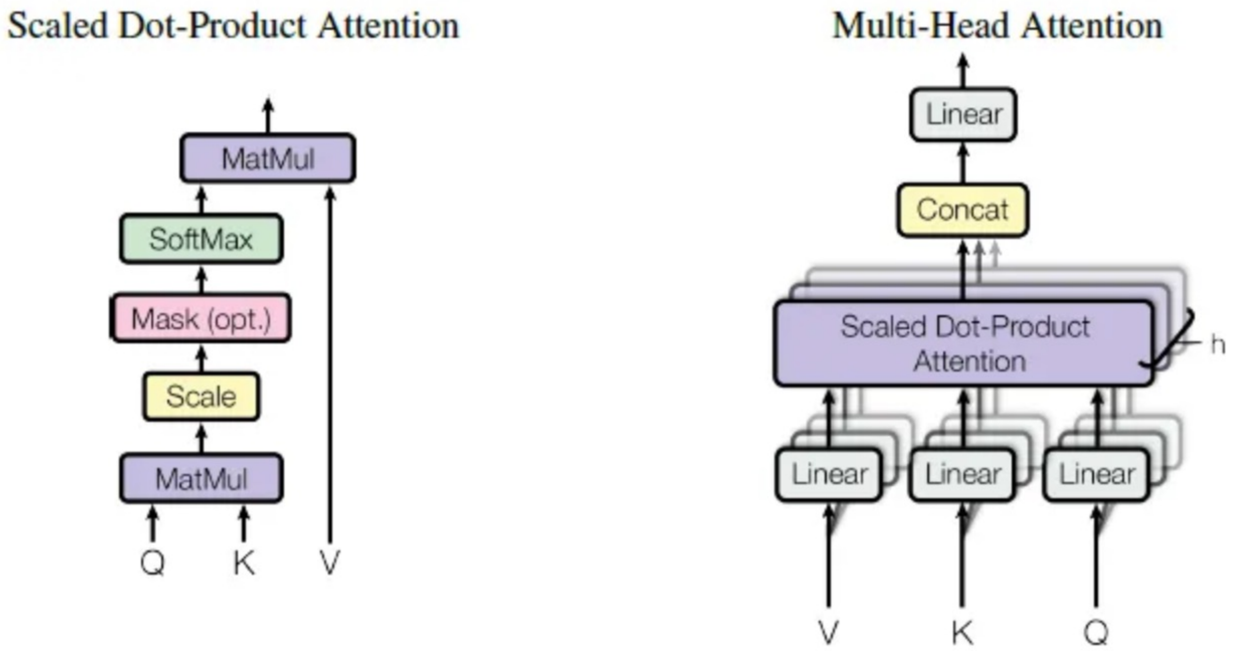
- One other nuance is that we scale the Softmax output.
- This is done to prevent the output being too close to 1 or too close to 0, which can lead to vanishing gradient issues.
- Here, sqrt(d_k) is square root of the dimensionality of the keys (in the original paper this is the same as the dimension of the word embeddings).
- We write this in big-letter notation because we can parallelize everything as very large matrix / tensor calculations.

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Multi-Head Attention

We have Multiple 'Heads' Paying Attention

- Each head can be thinking about something very different.
- By using multiple sets of query, key and value weight matrices, the network can learn to consider different combinations of input features.
- This lets it capture more complex patterns in the data.



From "Attention is all you need" paper by Vaswani, et al., 2017 [1]

Questions?