Introduction to Transformers

Navid Rekab-Saz

navid.rekabsaz@jku.at

Institute of Computational Perception

CP Lectures 22 Sep. 2020

JOHANNES KEPLER UNIVERSITY LINZ

Institute of Computational Perception

Agenda

- Background & Problem Definition
- Attention Mechanism
- Transformers

Agenda

- **Background & Problem Definition**
- Attention Mechanism
- Transformers

Notation

- $a \rightarrow$ scalar
- $\boldsymbol{b} \rightarrow$ vector
  - i^{th} element of $\boldsymbol{b}$ is the scalar b_i
- $\boldsymbol{C} \rightarrow$ matrix
  - i^{th} vector of $\boldsymbol{C}$ is $\boldsymbol{c}_i$
  - j^{th} element of the i^{th} vector of $\boldsymbol{C}$ is the scalar $c_{i,j}$
- Tensor: generalization of scalar, vector, matrix to any arbitrary dimension

Linear Algebra – Dot product

- $\boldsymbol{a} \cdot \boldsymbol{b}^T = c$
  - dimensions: $1 \times d \cdot d \times 1 = 1$

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix} = 5$$

- $\boldsymbol{a} \cdot \boldsymbol{B} = \boldsymbol{c}$
  - dimensions: $1 \times d \cdot d \times e = 1 \times e$

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 2 & 3 \\ 0 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 5 & 2 \end{bmatrix}$$

- $\boldsymbol{A} \cdot \boldsymbol{B} = \boldsymbol{C}$
  - dimensions: $l \times m \cdot m \times n = l \times n$

$$\begin{bmatrix} 1 & 2 & 3 \\ 1 & 0 & 1 \\ 0 & 0 & 5 \\ 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} 2 & 3 \\ 0 & 1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 5 & 2 \\ 3 & 2 \\ 5 & -5 \\ 8 & 13 \end{bmatrix}$$

- Linear transformation: dot product of a vector to a matrix

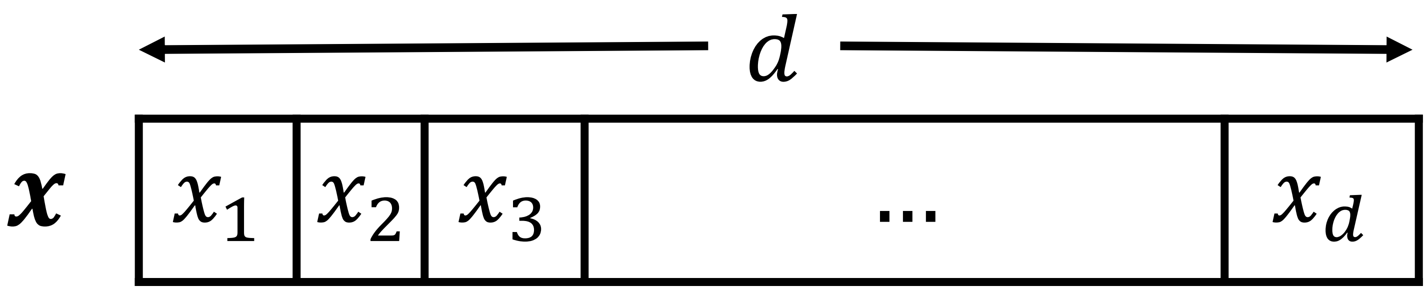
Probability

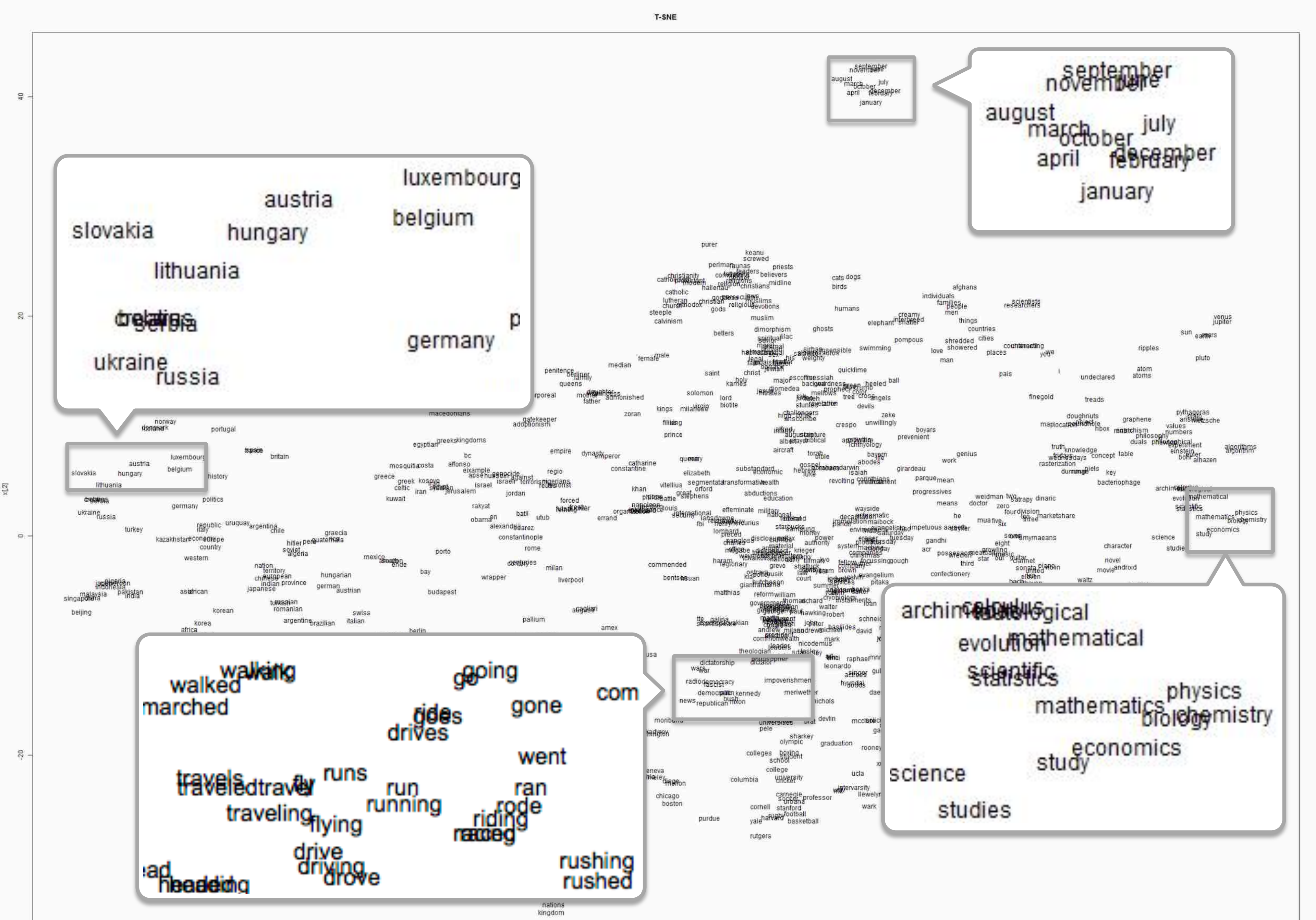
- Probability distribution
  - For a discrete random variable $\boldsymbol{z}$ with K states
    - $0 \leq p(z_i) \leq 1$
    - $\sum_{i=1}^{K} p(z_i) = 1$
  - E.g. with $K = 4$ states: $[0.2 \quad 0.3 \quad 0.45 \quad 0.05]$

Distributional Representation

- An entity is represented with a vector of d dimensions

- Distributed Representations
  - Each dimension (units) is a feature of the entity
  - Units in a layer are not mutually exclusive
  - Two units can be "active" at the same time





Word embeddings projected to a two-dimensional space

All we talk today is about …

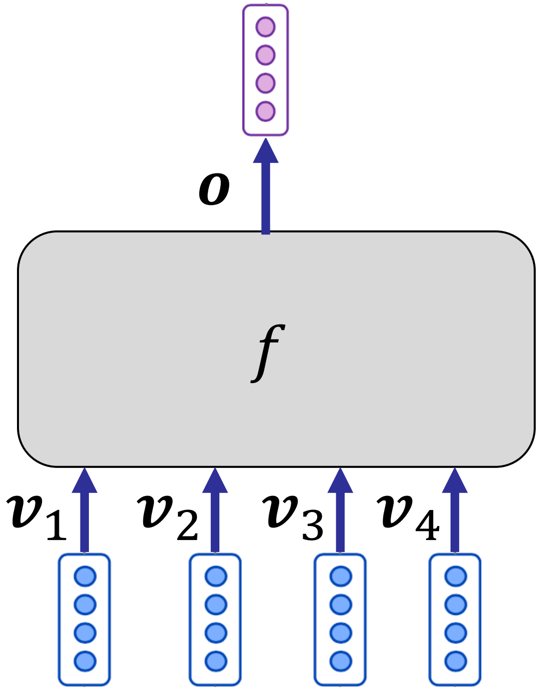
Compositional Representations

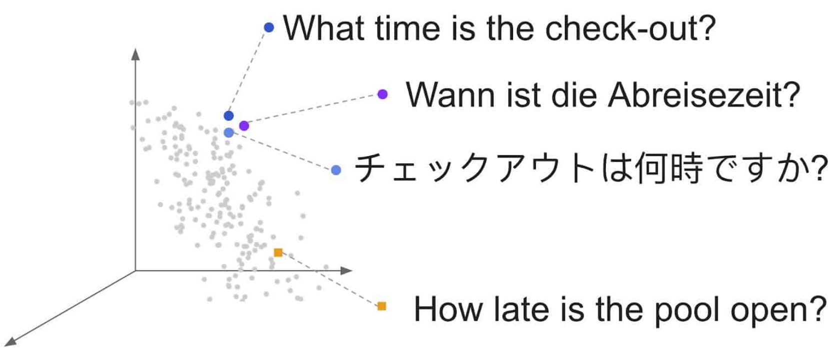
- Trying to address *representations composition* or *representations aggregation* problem

- Compositional representations appears in two scenarios:
  - Scenario 1: composing an output embedding from input embeddings
  - Scenario 2: contextualizing input embeddings

Compositional Representations

- **Scenario 1: composing an output embedding from input embeddings**
- Scenario 2: contextualizing input embeddings

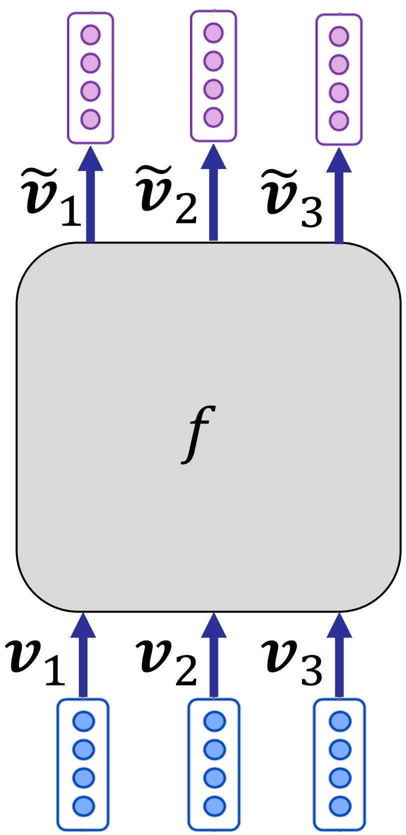


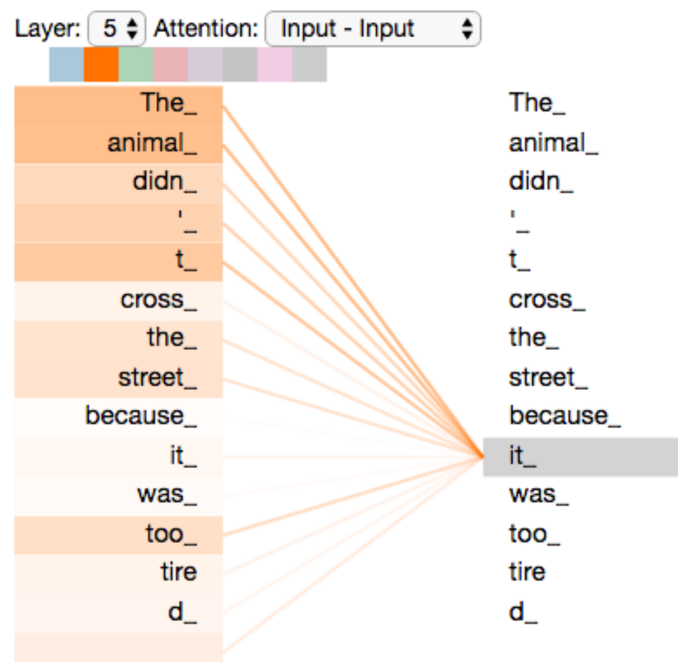


Source: https://megagon.ai/blog/emu-enhancing-multilingual-sentence-embeddings-with-semantic-similarity/

Compositional Representations

- Scenario 1: composing an output embedding from input embeddings
- **Scenario 2: contextualizing input embeddings**



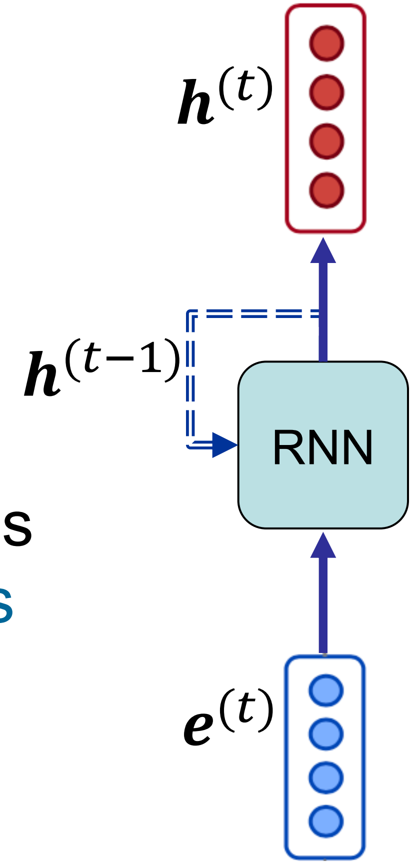


Recurrent Neural Networks – RECAP

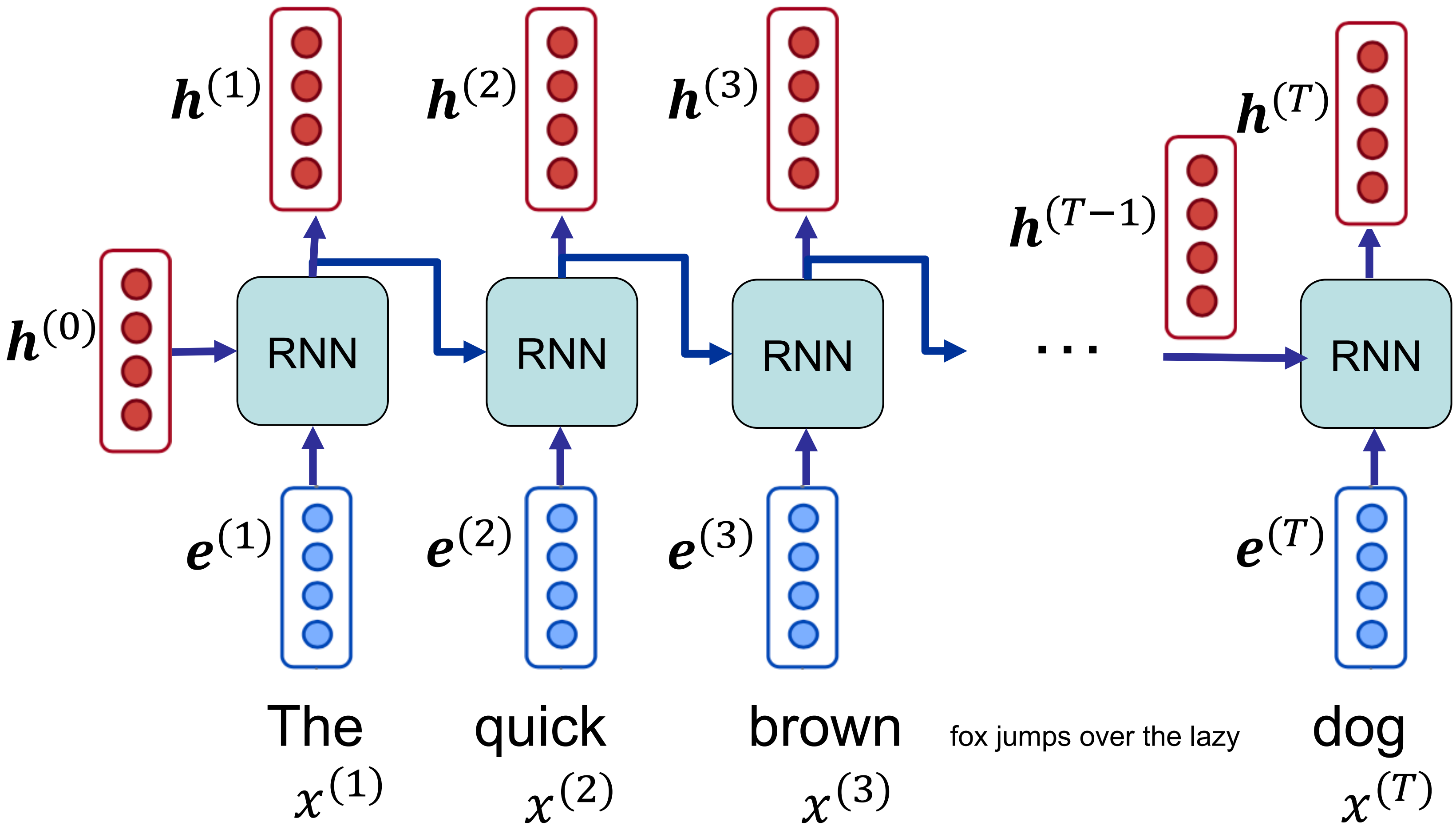
- Output $\boldsymbol{h}^{(t)}$ is a function of input $\boldsymbol{e}^{(t)}$ and the output of the previous time step $\boldsymbol{h}^{(t-1)}$

$$\boldsymbol{h}^{(t)} = \mathrm{RNN}(\boldsymbol{h}^{(t-1)}, \boldsymbol{e}^{(t)})$$

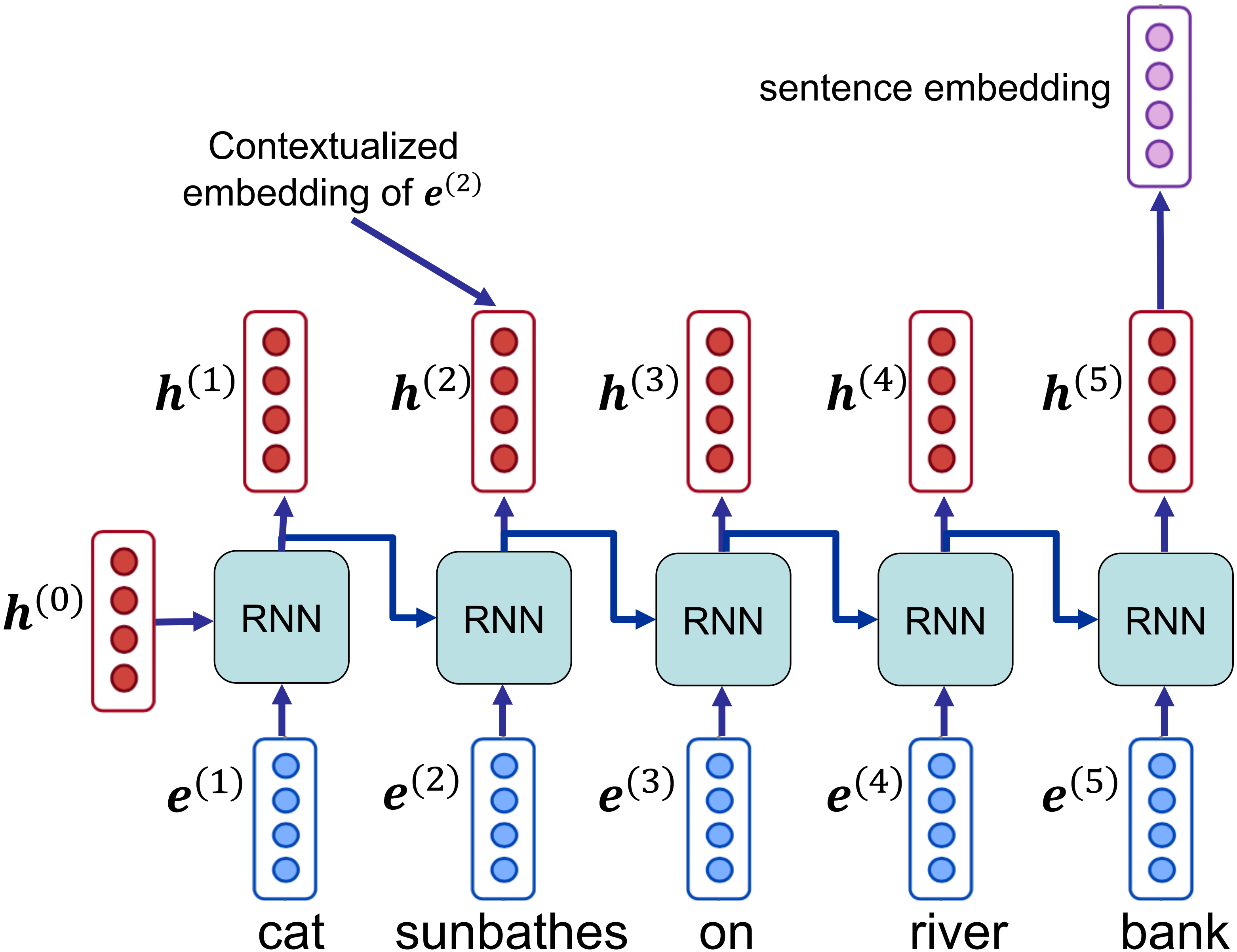
- $\boldsymbol{h}^{(t)}$ is called hidden state
- With hidden state $\boldsymbol{h}^{(t-1)}$, the model accesses to a sort of memory from all previous entities



RNN – Unrolling



RNN – Compositional embedding



Agenda

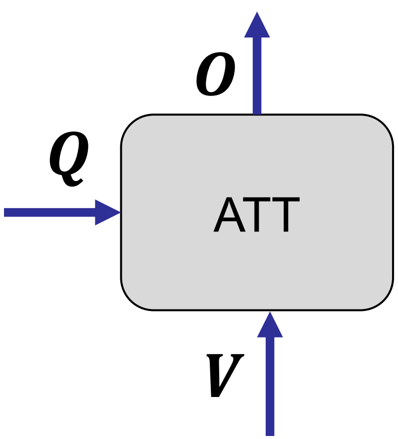
- Background & Problem Definition
- **Attention Mechanism**
- Transformers

Attention Networks

- Attention is a general Deep Learning method to
  - obtain a composed representation (output) …
  - from an arbitrary size of representations (values) …
  - depending on a given representation (query)

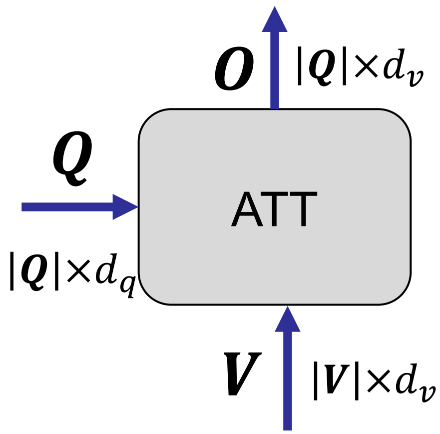
- General form of an attention network:

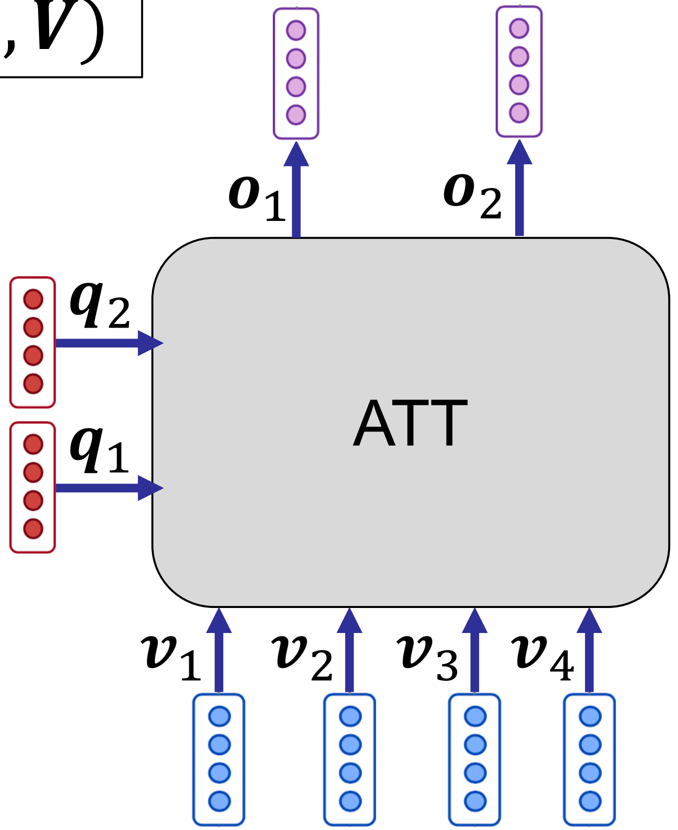
$$\boldsymbol{O} = \mathrm{ATT}(\boldsymbol{Q}, \boldsymbol{V})$$



Attention Networks

$$O = \mathrm{ATT}(Q, V)$$





- d_q, d_v are embedding dimensions of query and value vectors, respectively

We sometime say, each query vector $\boldsymbol{q}$ "*attends to*" the values

Attention Networks – definition

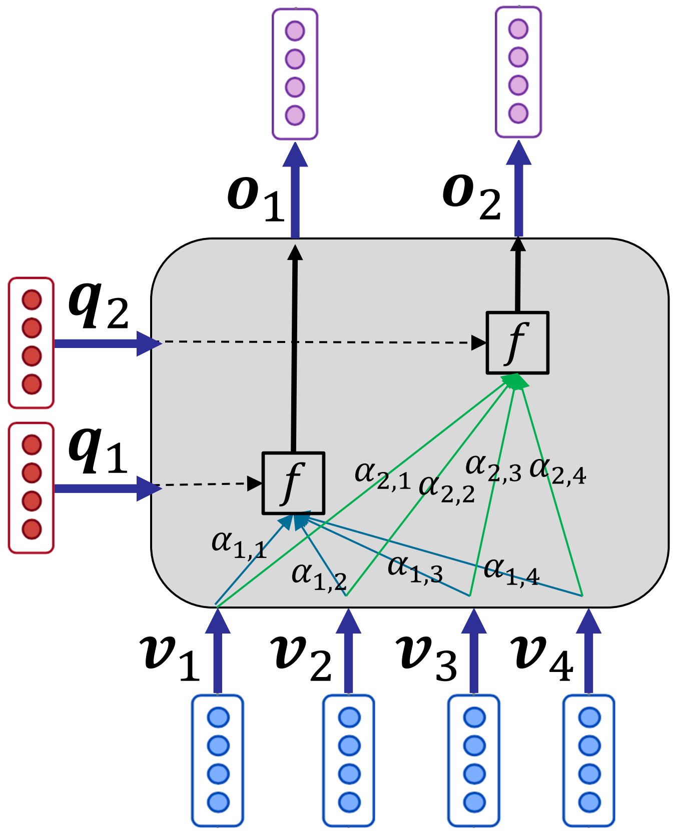
Formal definition:

- Given a set of vector *values* $\boldsymbol{V}$, and a set of vector *queries* $\boldsymbol{Q}$, **attention** is a technique to compute a weighted sum of the values, dependent on each query

- The weighted sum is a selective summary of the information contained in the values, where the query determines which values to focus on

- The weight in the weighted sum – for each query on each value – is called attention, and denoted by α

Attentions!



$\alpha_{i,j}$ is the attention of query $\boldsymbol{q}_i$ on value $\boldsymbol{v}_j$

$\boldsymbol{\alpha}_i$ is the vector of attentions of query $\boldsymbol{q}_i$ on value vectors $\boldsymbol{V}$

$\boldsymbol{\alpha}_i$ is a probability distribution

f is attention function

Attention Networks – formulation

- Given the query vector $\boldsymbol{q}_i$, an attention network assigns attention $\alpha_{i,j}$ to each value vector $\boldsymbol{v}_j$ using attention function f:

$$\alpha_{i,j} = f(\boldsymbol{q}_i, \boldsymbol{v}_j)$$

such that $\boldsymbol{\alpha}_i$ (vector of attentions for the ith query vector) forms a probability distribution

- The output regarding each query is the weighted sum of the value vectors (attentions as weights):

$$\boldsymbol{o}_i = \sum_{j=1}^{|\boldsymbol{V}|} \alpha_{i,j} \boldsymbol{v}_j$$

Attention variants

Basic dot-product attention

- First, non-normalized attention scores:

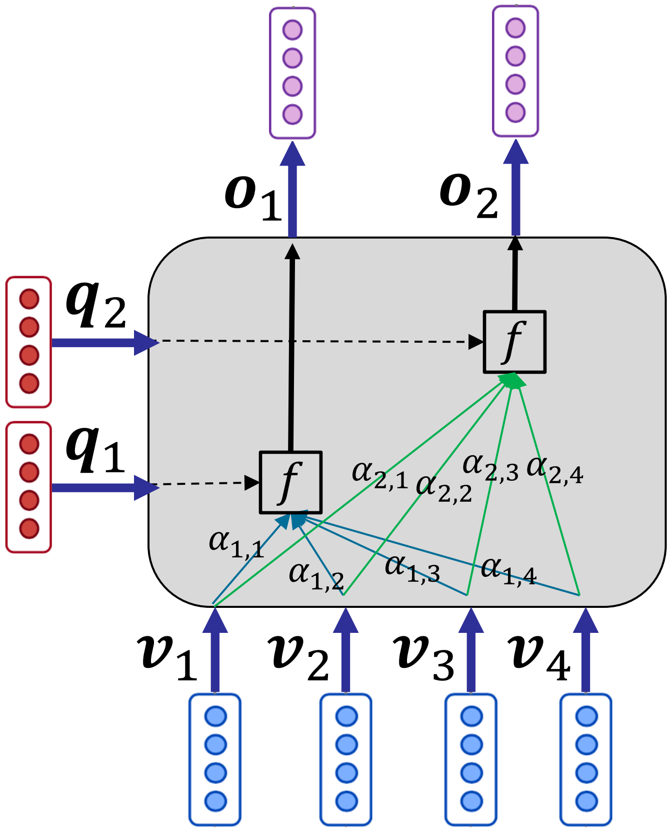
$$\tilde{\alpha}_{i,j} = \boldsymbol{q}_i \boldsymbol{v}_i^{\mathrm{T}}$$

  - In this variant $d_q = d_v$
  - There is no parameter to learn!

- Then, softmax over values:

$$\alpha_{i,j} = \mathrm{softmax}(\tilde{\boldsymbol{\alpha}}_i)_j$$

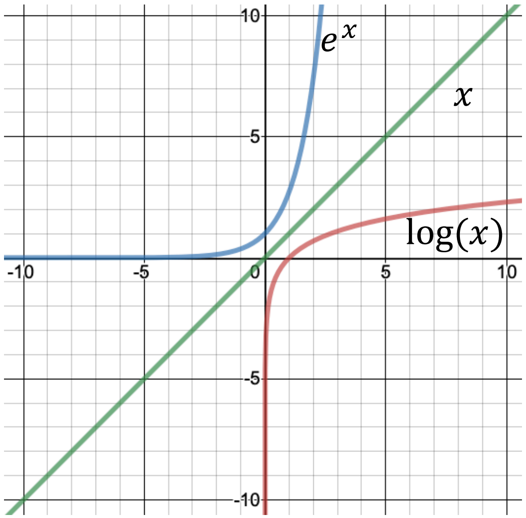
- Output (weighted sum): $\boldsymbol{o}_i = \sum_{j=1}^{|V|} \alpha_{i,j} \boldsymbol{v}_j$



softmax – RECAP

- softmax turns the vector to a probability distribution

$$\text{softmax}(\boldsymbol{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$



- Example with $K = 4$ classes

$$\boldsymbol{z} = \begin{bmatrix} 1 \\ 2 \\ 5 \\ 6 \end{bmatrix} \qquad \text{softmax}(\boldsymbol{z}) = \begin{bmatrix} 0.004 \\ 0.013 \\ 0.264 \\ 0.717 \end{bmatrix}$$

Attention variants

Multiplicative attention

- First, non-normalized attention scores:

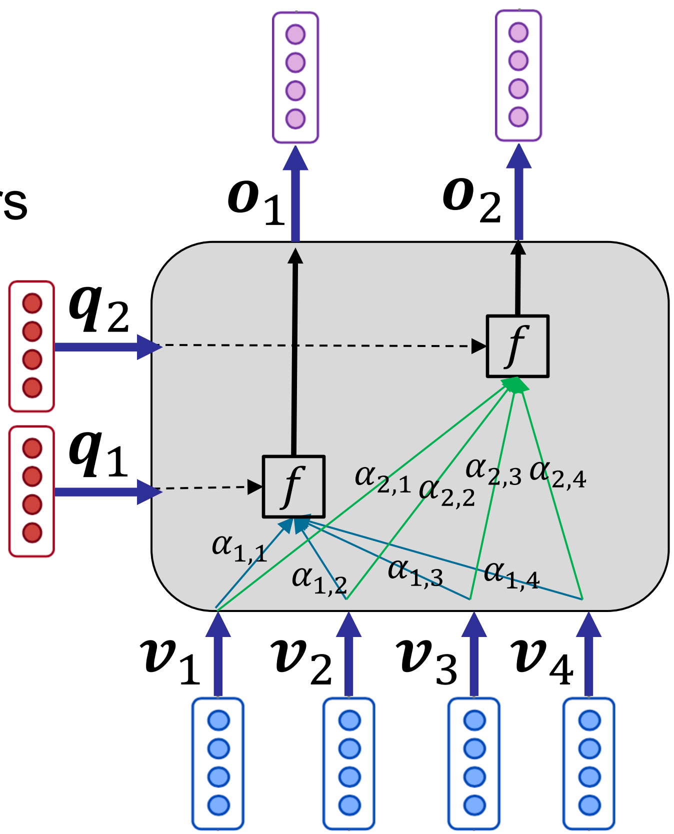
$$\tilde{\alpha}_{i,j} = \boldsymbol{q}_i \boldsymbol{W} \boldsymbol{v}_i^{\mathrm{T}}$$

  - $\boldsymbol{W}$ is a matrix of model parameters
  - provides a linear function for measuring relations between query and value vectors

- Then, softmax over values:

$$\alpha_{i,j} = \mathrm{softmax}(\tilde{\boldsymbol{\alpha}}_i)_j$$

- Output (weighted sum): $\boldsymbol{o}_i = \sum_{j=1}^{|V|} \alpha_{i,j} \boldsymbol{v}_j$



Attention variants

Additive attention

- First, non-normalized attention scores:

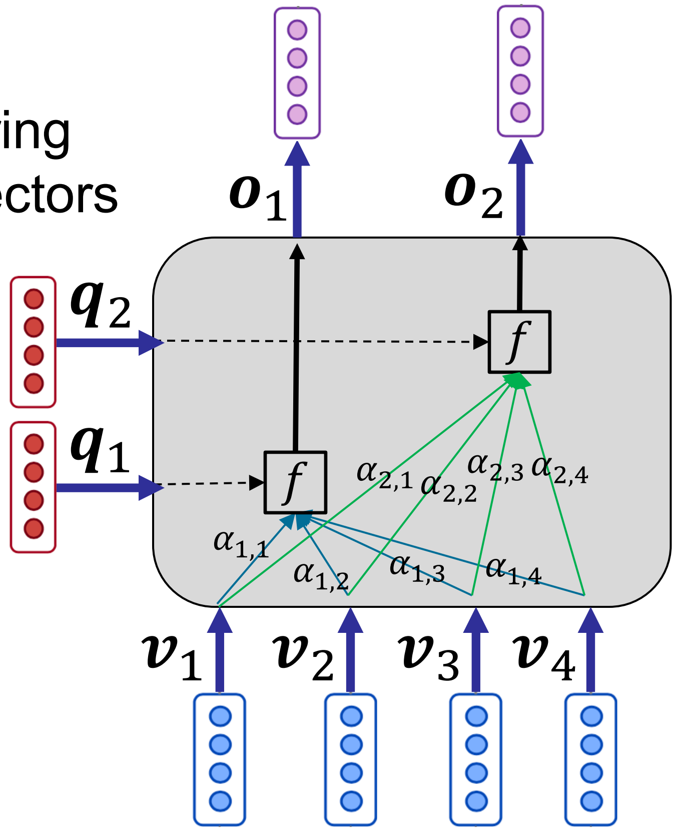
$$\tilde{\alpha}_{i,j} = \boldsymbol{u}^{\mathrm{T}} \tanh(\boldsymbol{q}_i \boldsymbol{W}_1 + \boldsymbol{v}_j \boldsymbol{W}_2)$$

  - $\boldsymbol{W}_1$, $\boldsymbol{W}_2$, and $\boldsymbol{u}$ are model parameters
  - provides a non-linear function for measuring relations between the query and value vectors

- Then, softmax over values:

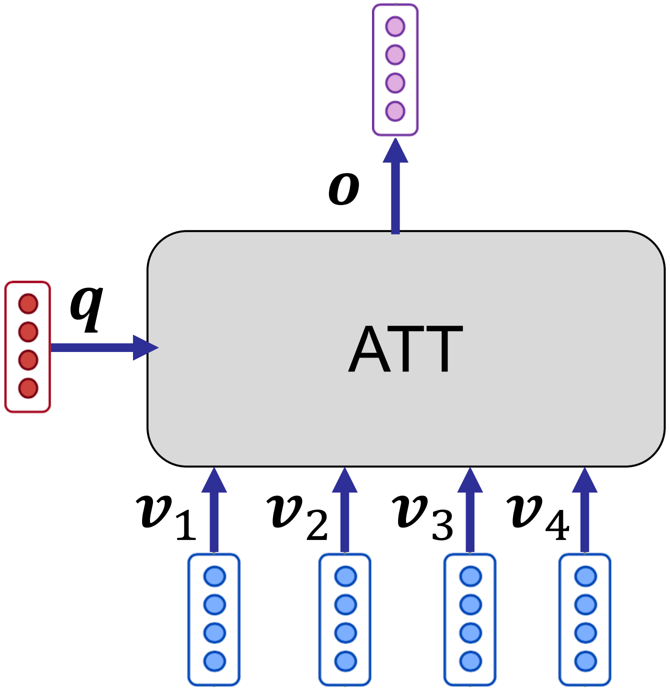
$$\alpha_{i,j} = \mathrm{softmax}(\tilde{\boldsymbol{\alpha}}_i)_j$$

- Output (weighted sum): $\boldsymbol{o}_i = \sum_{j=1}^{|V|} \alpha_{i,j} \boldsymbol{v}_j$



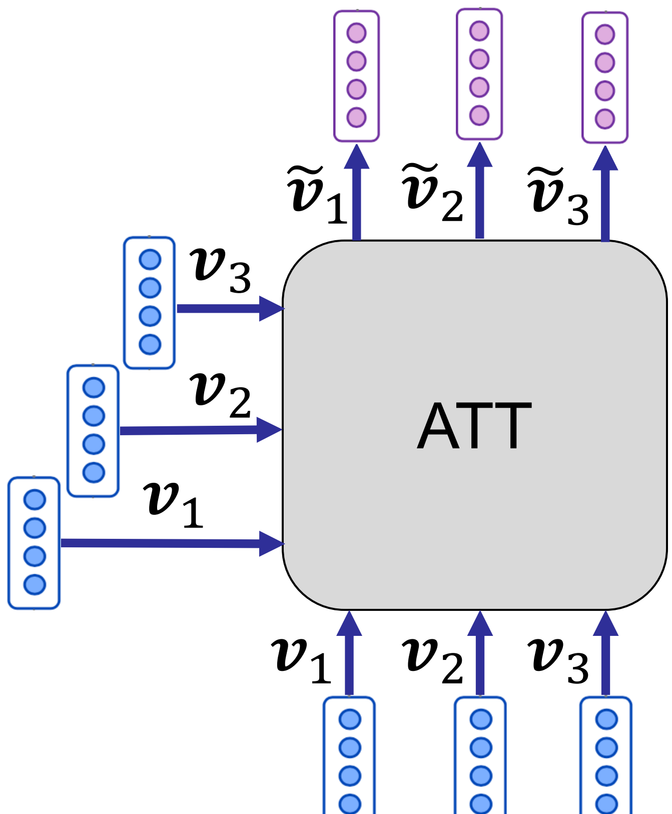
Attention in practice

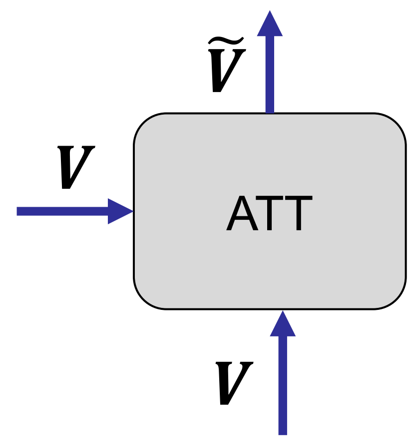
- Attention is used to create a compositional embedding of value vectors, according to a query
  - E.g. in document classification
    - Where values are document's word vectors, and query is a parameter vector



Self-attention

- In self-attention, values are the same as queries: $\boldsymbol{Q} = \boldsymbol{V}$
- Mainly used to encode a sequence $\boldsymbol{V}$ to another sequence $\widetilde{\boldsymbol{V}}$
- Each encoded vector is a contextual embedding of the corresponding input vector
  - $\widetilde{\boldsymbol{v}}_i$ is the contextual embedding of $\boldsymbol{v}_i$





Attention – summary

- Attention is a way to focus on particular parts of the input, and create a compositional embedding

- It is done by defining an attention distribution over inputs, and calculating their weighted sum

- A more generic definition of attention network has two inputs: key vectors $\boldsymbol{K}$, and value vectors $\boldsymbol{V}$
  - Key vectors are used to calculate attentions
  - and, as before, output is the weighted sum of value vectors
  - In practice, in most cases $\boldsymbol{K} = \boldsymbol{V}$. So we consider our (slightly simplified) definition in most parts of this course

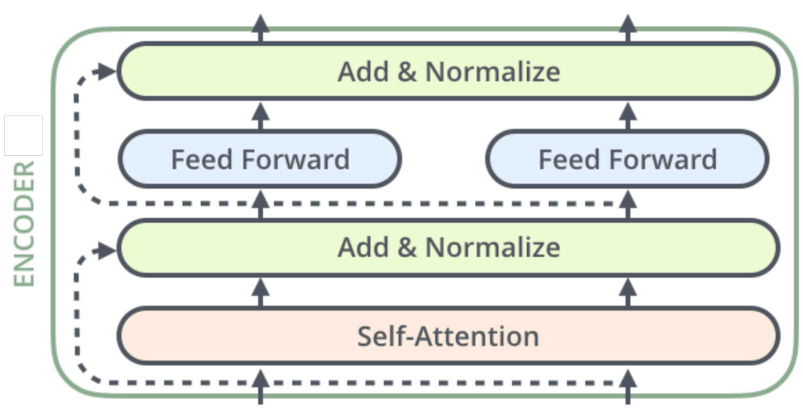
Agenda

- Background & Problem Definition
- Attention Mechanism
- **Transformers**

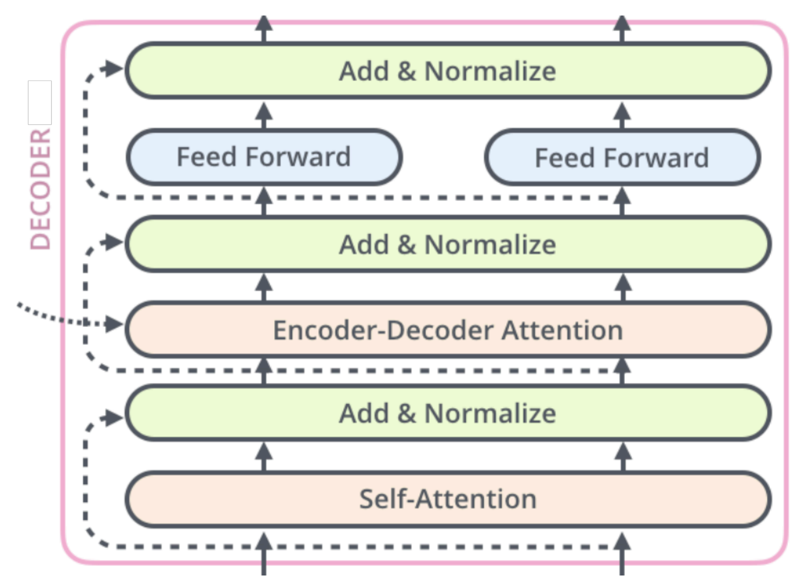
Transformers

- An attention model with DL best practices!
- Originally introduced for machine translation, and now widely adopted for non-recurrent sequence encoding and decoding

Transformer encoder



Transformer decoder



Attention is all you need. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Polosukhin, I. (2017). In *NeurIPS*.

Transformer encoder

Transformer encoder consists of two sub-layers:

- 1st : Multi-head scaled dot-product self-attention
- 2nd : Position-wise feed forward
- Each sub-layer is followed by layer normalization and residual networks … and drop-outs are applied after each computation

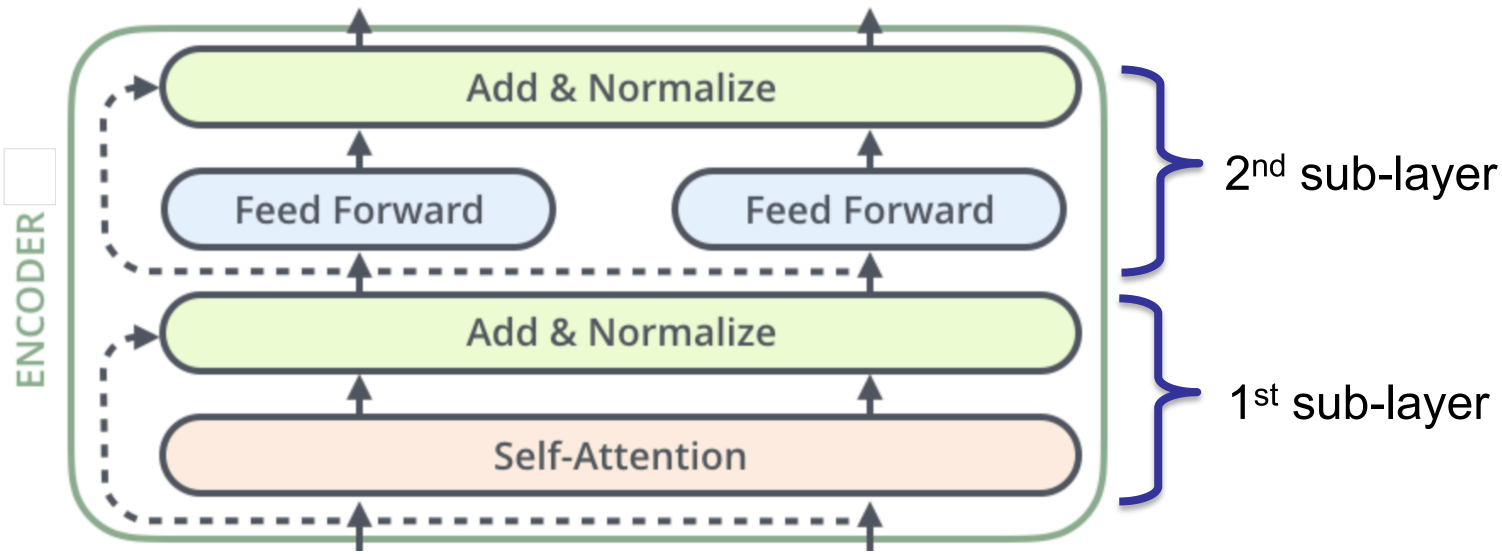


Figure source: http://jalammar.github.io/illustrated-transformer/

Transformer encoder

Let's start from multi-head scaled dot-product self-attention:

1. Scaled dot-product attention
2. Multi-head attention
3. self-attention (recap)

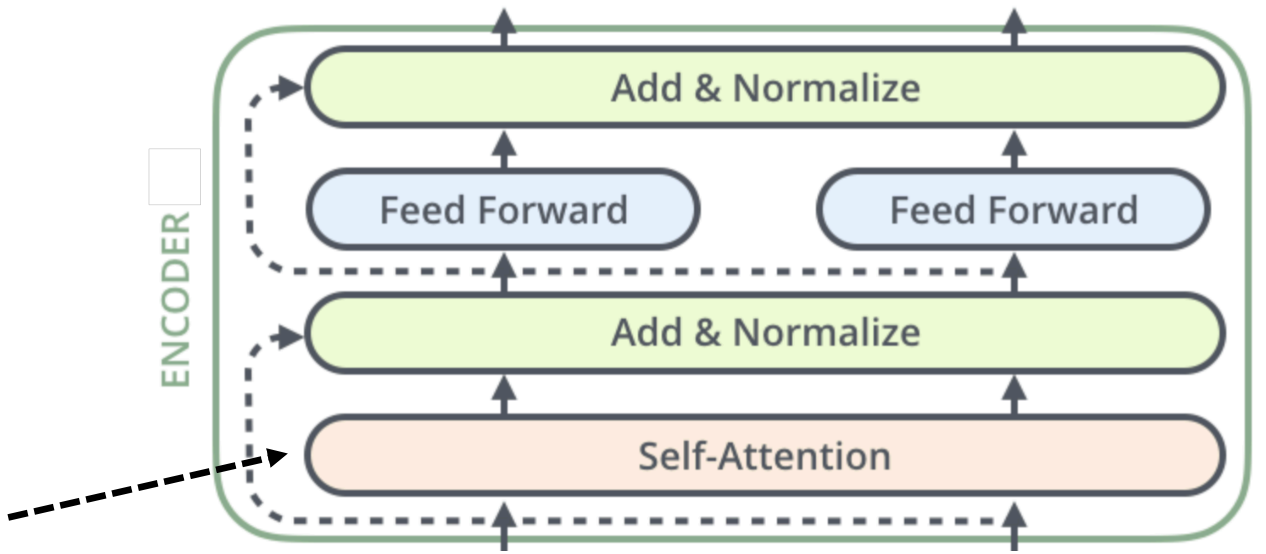


Figure source: http://jalammar.github.io/illustrated-transformer/

Recap: basic dot-product attention

- First, non-normalized attention scores:

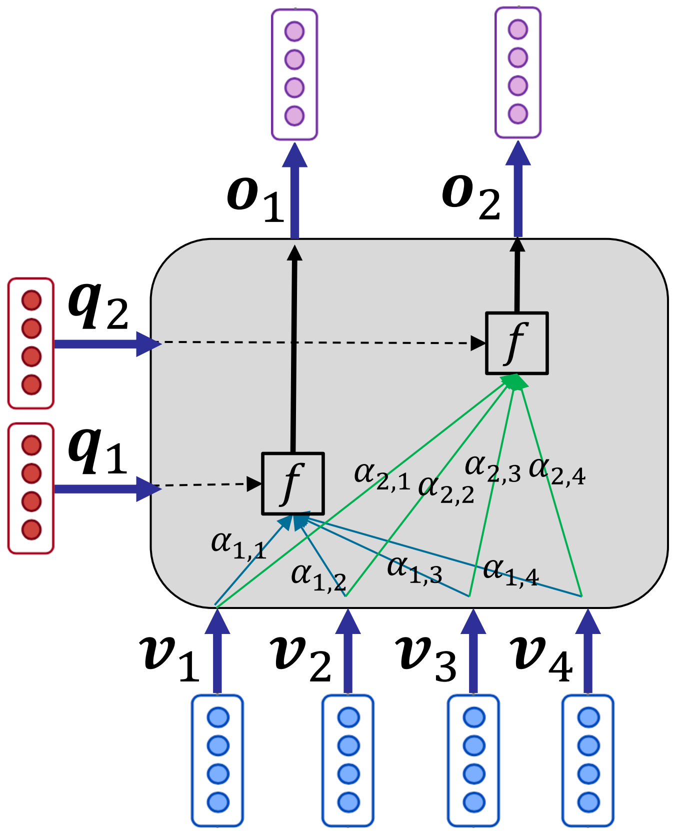
$$\tilde{\alpha}_{i,j} = \boldsymbol{q}_i \boldsymbol{v}_j^{\mathrm{T}}$$

  - $d = d_q = d_v$ dimension of vectors
  - has no parameter!

- Then, softmax over values:

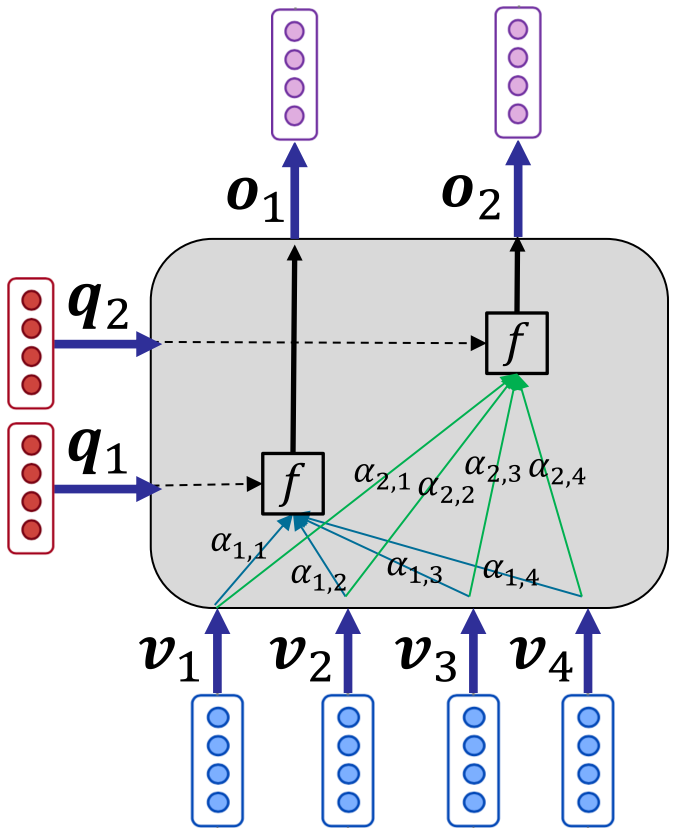
$$\alpha_{i,j} = \mathrm{softmax}(\tilde{\boldsymbol{\alpha}}_i)_j$$

- Output (weighted sum): $\boldsymbol{o}_i = \sum_{j=1}^{|V|} \alpha_{i,j} \boldsymbol{v}_j$



Scaled dot-product attention

- Problem with basic doc-product attention:
  - As d gets large, the variance of $\tilde{\alpha}_{i,j}$ increases …
  - … this makes softmax very peaked for some values $\tilde{\boldsymbol{\alpha}}_i$ …
  - … and hence its gradient gets smaller
- Solution: normalize/scale $\tilde{\alpha}_{i,j}$ by size of d



<u>Scaled dot-product attention</u>

- Non-normalized attention scores:

$$\tilde{\alpha}_{i,j} = \frac{\boldsymbol{q}_i \boldsymbol{v}_j^{\mathrm{T}}}{\sqrt{d}}$$

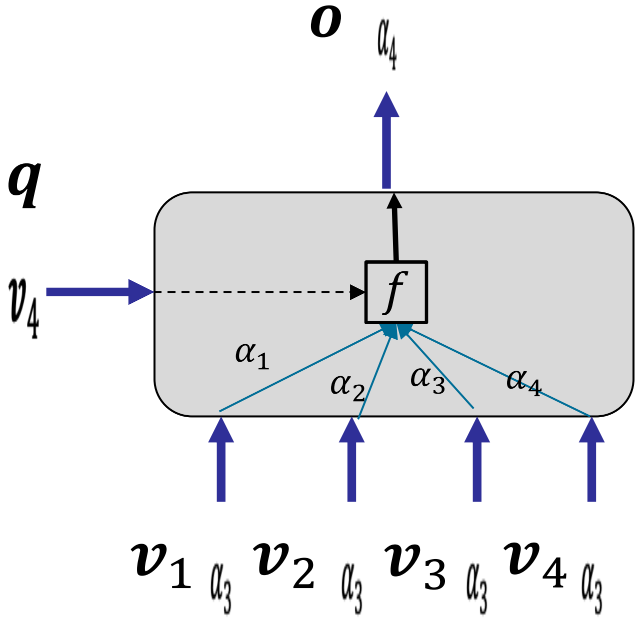
- Softmax over values: $\alpha_{i,j} = \mathrm{softmax}(\tilde{\boldsymbol{\alpha}}_i)_j$

- Output (weighted sum): $\boldsymbol{o}_i = \sum_{j=1}^{|\boldsymbol{V}|} \alpha_{i,j} \boldsymbol{v}_j$

Problem with (single-head) attention

- In all attention networks so far, the final attention of query $\boldsymbol{q}$ on value vectors $\boldsymbol{V}$ are normalized with softmax
  - Recall that softmax makes the maximum value much higher than the other

$$\boldsymbol{z} = [1 \quad 2 \quad 5 \quad 6] \rightarrow \text{softmax}(\boldsymbol{z}) = [0.004 \quad 0.013 \quad 0.264 \quad 0.717]$$



- Common in language, a word may be related to several other words in sequence, each through a specific concept
  - Like the relations of a verb to its subject and to its object
- However in a (single-head) attention network, all concepts are aggregated in one attention set
- Due to softmax, value vectors must compete for the attention of query vector → softmax bottleneck

Multi-head attention

- Multi-head attention approaches this by calculating multiple sets of attentions between queries and values

Multi-head attention:

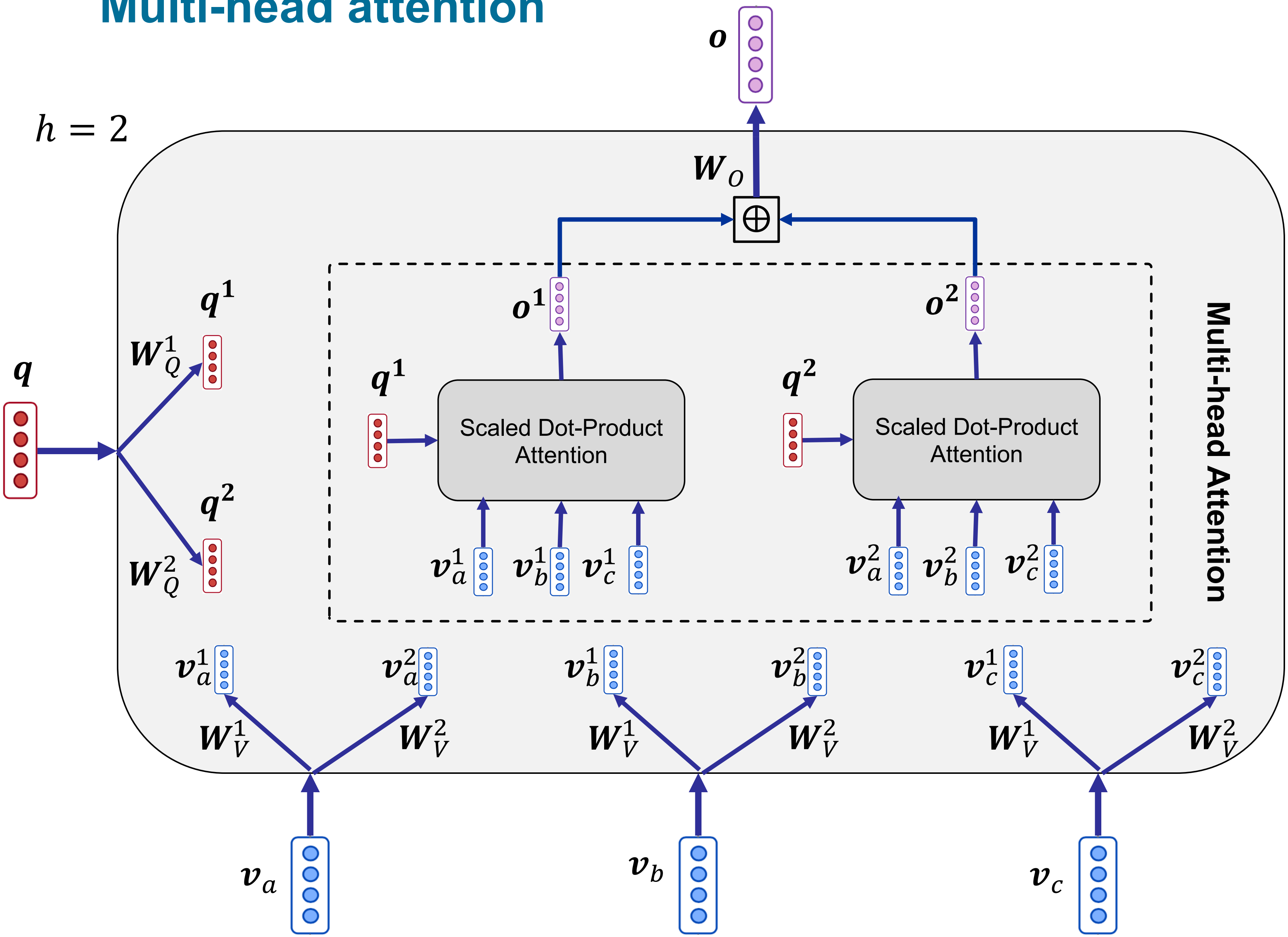
1. Down-project query and value vectors to h subspaces (heads)

2. In each subspace, calculate a simple attention network using the queries and values projected in the subspace, resulting in output vectors of the subspace

3. Concatenate the output vectors of all subspaces regarding each query, resulting in the final output of each query

- In multi-head attention, each head (and each subspace) can specialize on capturing a specific kind of relations

Multi-head attention



Multi-head attention – formulation

- Down-project every query $\boldsymbol{q}_i$ to h vectors, each with size ${}^{d}/_{h}$:

size: ${}^{d}/_{h}$

$$\boldsymbol{q}_i^1 = \boldsymbol{q}_i \boldsymbol{W}_Q^1 \quad \dots \quad \boldsymbol{q}_i^h = \boldsymbol{q}_i \boldsymbol{W}_Q^h$$

Matrix size: $d \times {}^{d}/_{h}$

- Down-project every value $\boldsymbol{v}_j$ to h vectors, each with size ${}^{d}/_{h}$:

size: ${}^{d}/_{h}$

$$\boldsymbol{v}_j^1 = \boldsymbol{v}_j \boldsymbol{W}_V^1 \quad \dots \quad \boldsymbol{v}_j^h = \boldsymbol{v}_j \boldsymbol{W}_V^h$$

Matrix size: $d \times {}^{d}/_{h}$

- Calculate outputs of subspaces corresponding to $\boldsymbol{q}_i$:

size: ${}^{d}/_{h}$

$$\boldsymbol{o}_i^1 = \mathrm{ATT}(\boldsymbol{q}_i^1, \boldsymbol{V}^1) \quad \dots \quad \boldsymbol{o}_i^h = \mathrm{ATT}(\boldsymbol{q}_i^h, \boldsymbol{V}^h)$$

- Concatenate outputs of subspaces for $\boldsymbol{q}_i$ as its final output:

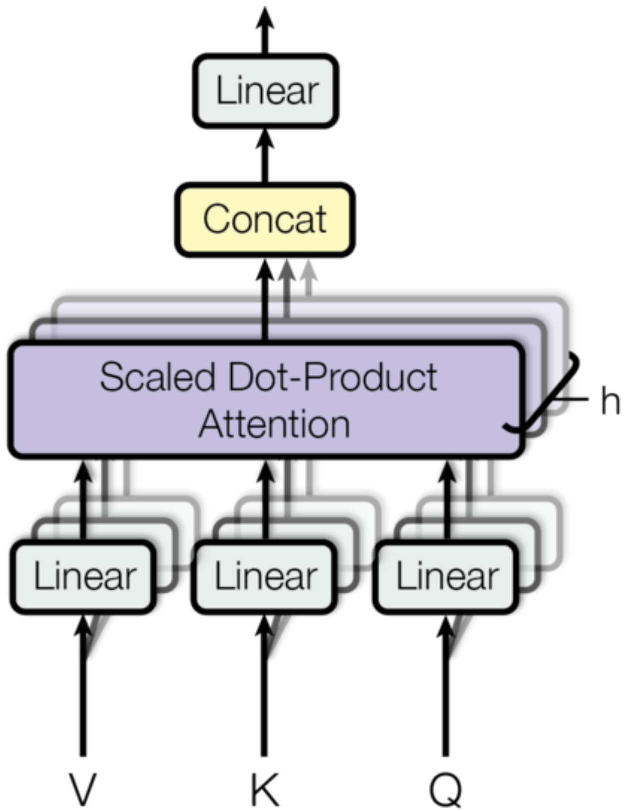
size: d

$$\boldsymbol{o}_i = \boldsymbol{W}_O \, [\boldsymbol{o}_i^1; \dots; \boldsymbol{o}_i^h]$$

Size: $d \times d$
This matrix linearly combines the dimensions of the concatenated vectors

Parameters are shown in red

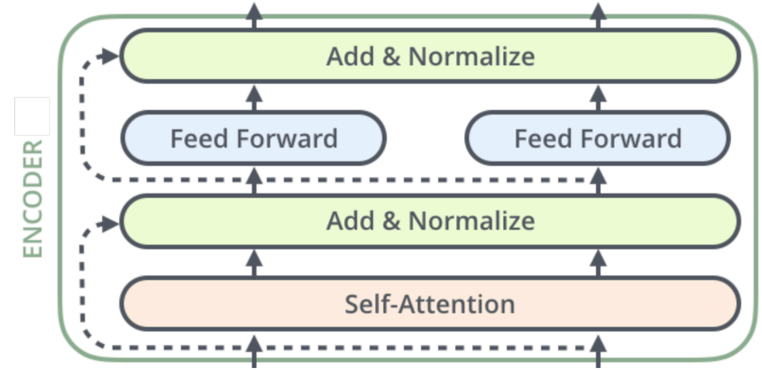
Multi-head attention – in original paper

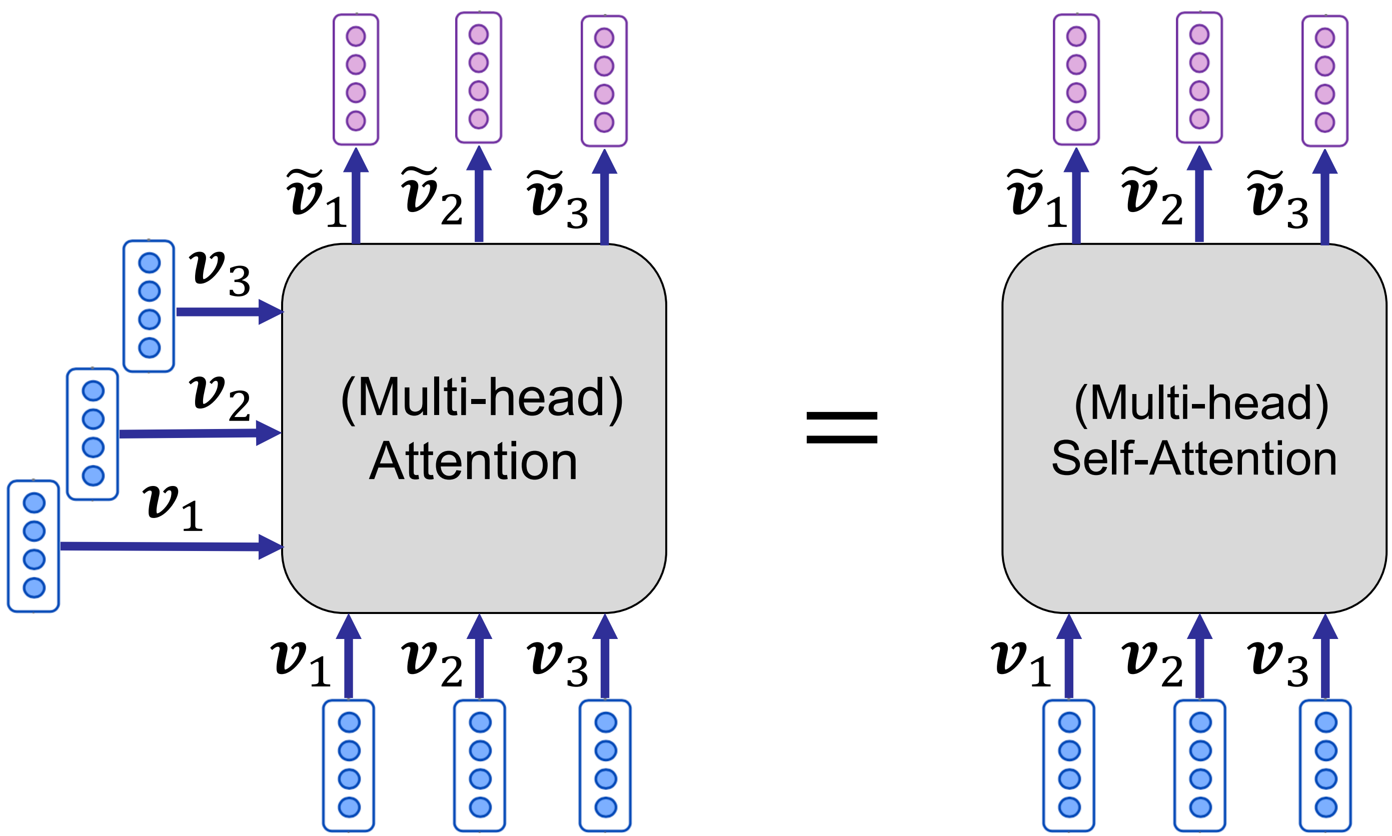


- Default number of heads in Transformers: $h = 8$
- Recall: Attentions (and Transformers) in fact have three inputs (not two), namely queries, keys, and values.
  - Keys are used to calculate attentions
  - Values are used to produce outputs

Self-attention – recap

- Values are the same as queries
- Each encoded vector is the contextual embedding of the corresponding input vector
    - $\tilde{v}_i$ is the contextual embedding of v_i



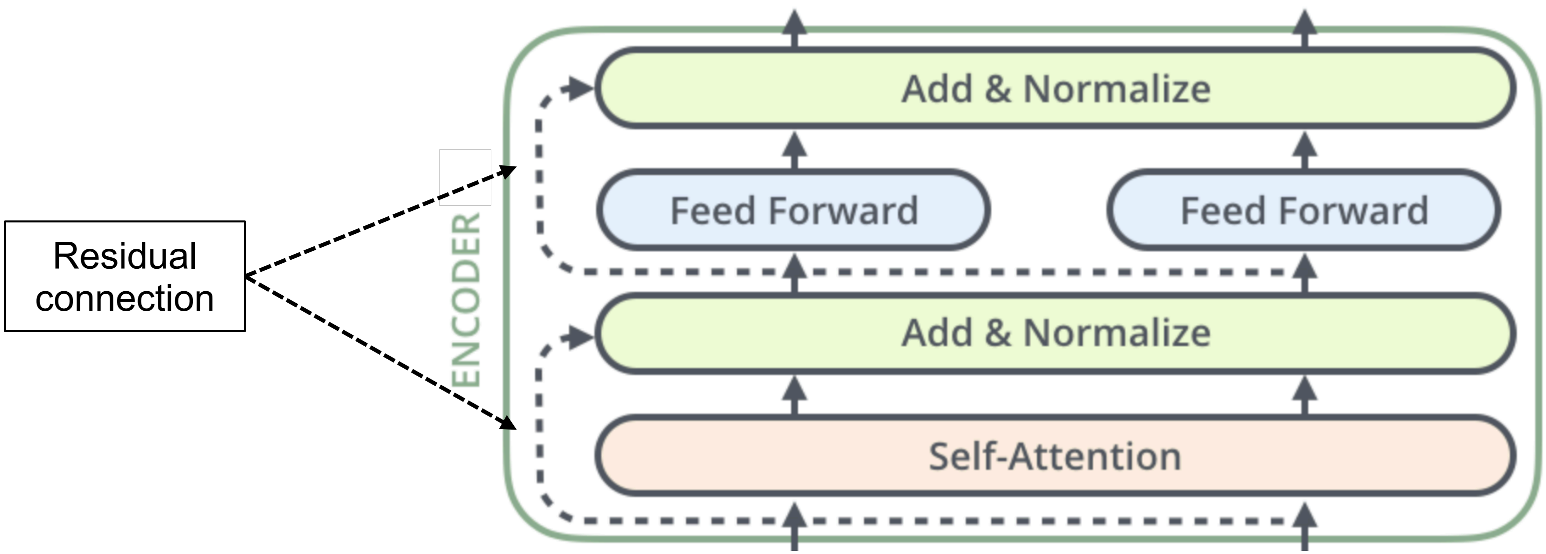


Residuals

- Residual (short-cut) connection:

$$\text{output} = f(x) + x$$

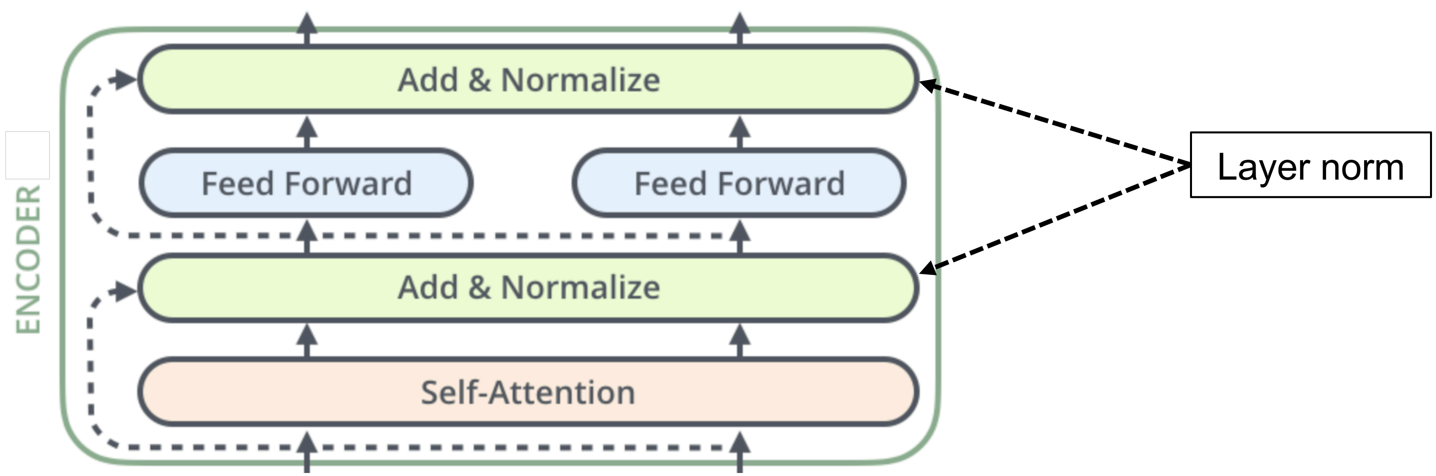
- Learn in detail:
  - He, Kaiming; Zhang, Xiangyu; Ren, Shaoqing; Sun, Jian (2016). "Deep Residual Learning for Image Recognition" . In proc. of CVPR
  - Srivastava, Rupesh Kumar; Greff, Klaus; Schmidhuber, Jürgen (2015). "Highway Networks". https://arxiv.org/pdf/1505.00387.pdf



Layer normalization

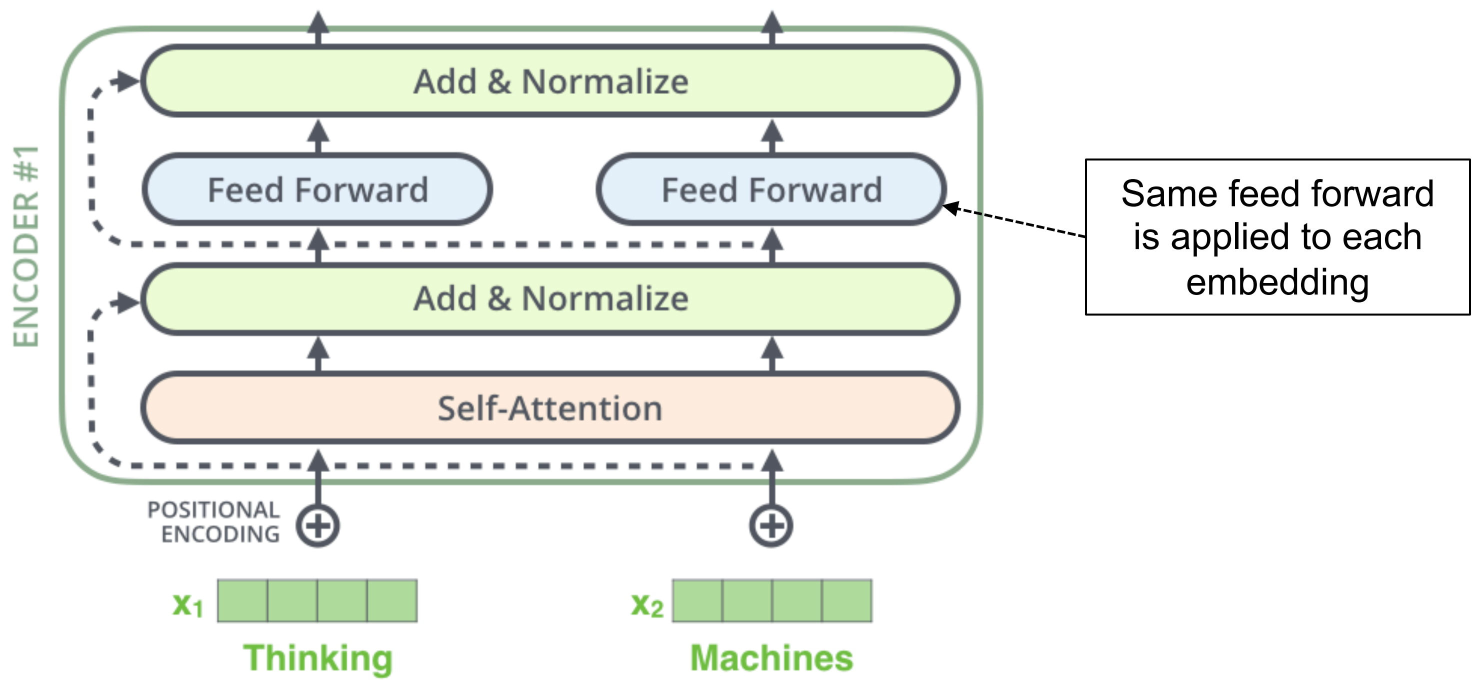
- Layer normalization changes each vector to have mean 0 and variance 1 …
  - … and learns two more parameter vectors per layer that set new means and variances for each dimension of the vectors

- Learn in detail:
  - Batch Normalization: Deep Learning book section 8.7.1 http://www.deeplearningbook.org/contents/optimization.html
  - Talk by Goodfellow https://www.youtube.com/watch?v=Xogn6veSyxA&feature=youtu.be
  - Paper: https://arxiv.org/pdf/1607.06450.pdf



Feed Forward on embedding

- In Transformers, a two-layer feed forward neural network (with ReLU) is applied to each embedding
  - With the feed forward network, the Transformers gain the capacity to learn non-linear transformations over each (contextualized) embedding



Transformer encoder – summary

- Multi-head self-attention model followed by a feed-forward layer

Benefits (as in attentions)

- No locality bias
  - A long-distance context has "*equal opportunity*"
- Single computation per layer (non-autoregressive)
  - Friendly with high parallel computations of GPU

- Look here for self-teaching and the PyTorch implementation:
  - http://nlp.seas.harvard.edu/2018/04/03/attention.html
  - also available on Google Colab

Finito!

Position embeddings

- Transformers are agnostic regarding the position of tokens (no locality bias)
  - A context token in long-distance has the same effect as one in short-distance
- However, the positions of tokens can still bring useful information

Position embeddings – a common solution in Transformers:

- Consider an embedding for each position, and add its values to the token embedding at that position
  - Position embedding is usually created using a sine/cosine function, or learned end-to-end with the model
  - Using position embeddings, the same word at different locations will have different overall representations

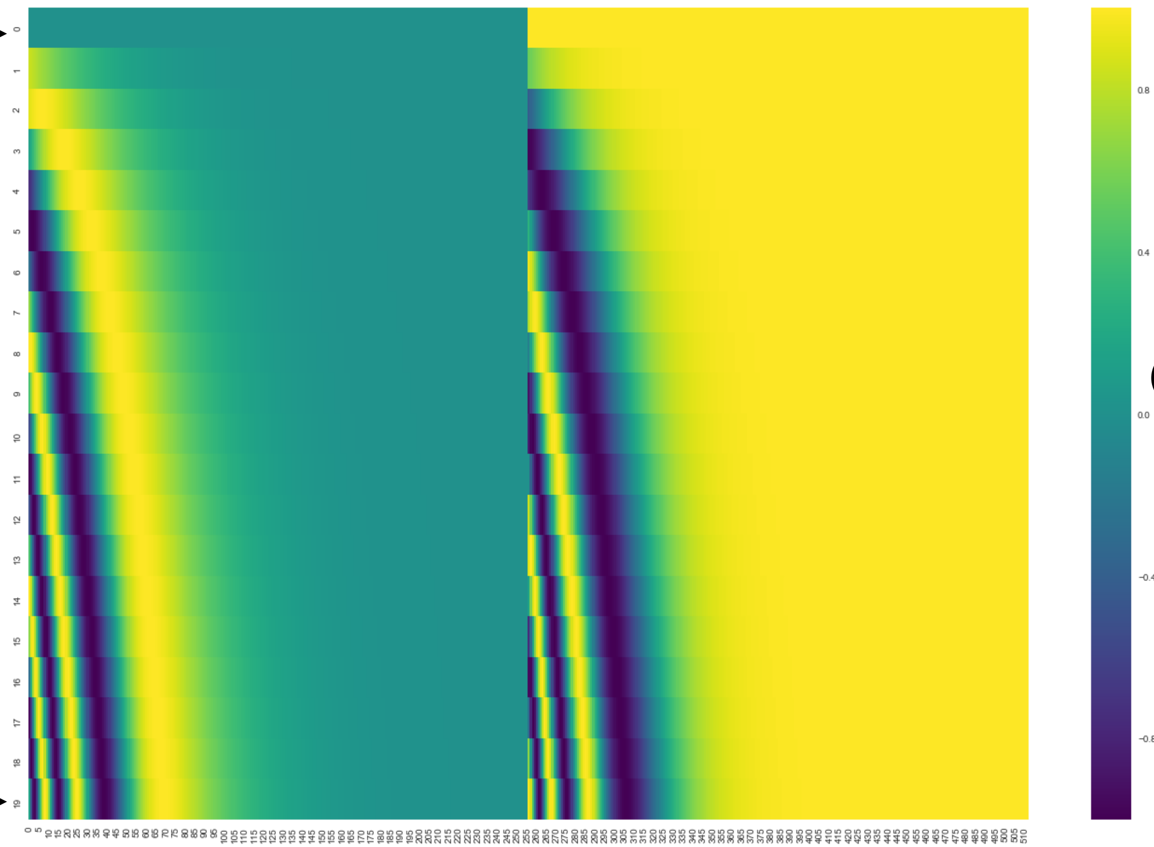
Position embeddings – examples

An example of embeddings with four dimensions:

Position embedding for location 0

Position embeddings

Position embedding for location 20



Values from -1 (dark) to +1 (light)

Dimensions (512)

Source: http://jalammar.github.io/illustrated-transformer/