Machine Learning et Intelligence Artificielle
Introduction au Deep Learning

Sorbonne Université
Patrick Gallinari, patrick.gallinari@sorbonne-universite.fr
Olivier Schwander, olivier.schwander@sorbonne-universite.fr
04/2026

Course Outline and Organization

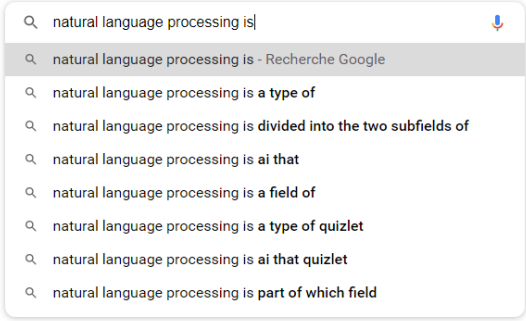
- Introductory course on Neural Networks and Deep Learning
- Organization
  - 2x3 h course, 2x3 h practice
- Outline
  - Neural Networks and Deep Learning
    - Introductory Concepts – Perceptron - Adaline
    - Computation Graph
    - Multilayer Perceptrons
    - Convolutional Neural Networks – Vision applications
    - Language models - Transformers

Language models

Language models

- Objective:
  - Probability models of sequences $(x^1, x^2, \ldots, x^t)$
  - Items may be words, characters, character ngrams, word pieces, etc
  - Formally: given a sequence of items, what is the probability of the next item?
    - $p(x^t | x^{t-1}, \ldots, x^1)$
- Example
  - « S'il vous plaît… dessine-moi …» what next ?
  - « $x^1 x^2 x^3$ … … … … … … … … x^{t-1} … » what is x^t ?
- Language models in everyday use
  - Sentence completion
    - Search engine queries
    - Smartphone messages, etc
  - Speech recognition, handwriting recognition, etc

Google



Language models

- Language models can be used to compute the probability of a piece of text
- Let $(x^1, x^2, \ldots, x^{\mathrm{T}})$ be a sequence of text, its probability according to a language model is:
  - $p(x^1, x^2, \ldots, x^{\mathrm{T}}) = \prod_{t=1}^{T} p(x^t | x^{t-1}, \ldots, x^1)$
    - With $p(x^t | x^{t-1}, \ldots, x^1)$ computed by the language model

Language models
How to learn a language model - n-grams

- A simple solution: n-grams
  - n-grams are sequences of n consecutive words (or characters, or any items)
  - Language model is based on n-gram statistics
  - Markov assumption
    - x^t only depends on the $n-1$ preceding words
      - $p(x^t|x^{t-1},\dots,x^1)=p(x^t|x^{t-1},\dots,x^{t-n+1})$
    - Use Bayes formula $p(x^t|x^{t-1},\dots,x^{t-n+1}) = \frac{p(x^t,x^{t-1},\dots,x^{t-n+1})}{p(x^{t-1},\dots,x^{t-n+1})}$

n-gram probability

n-1-gram probability

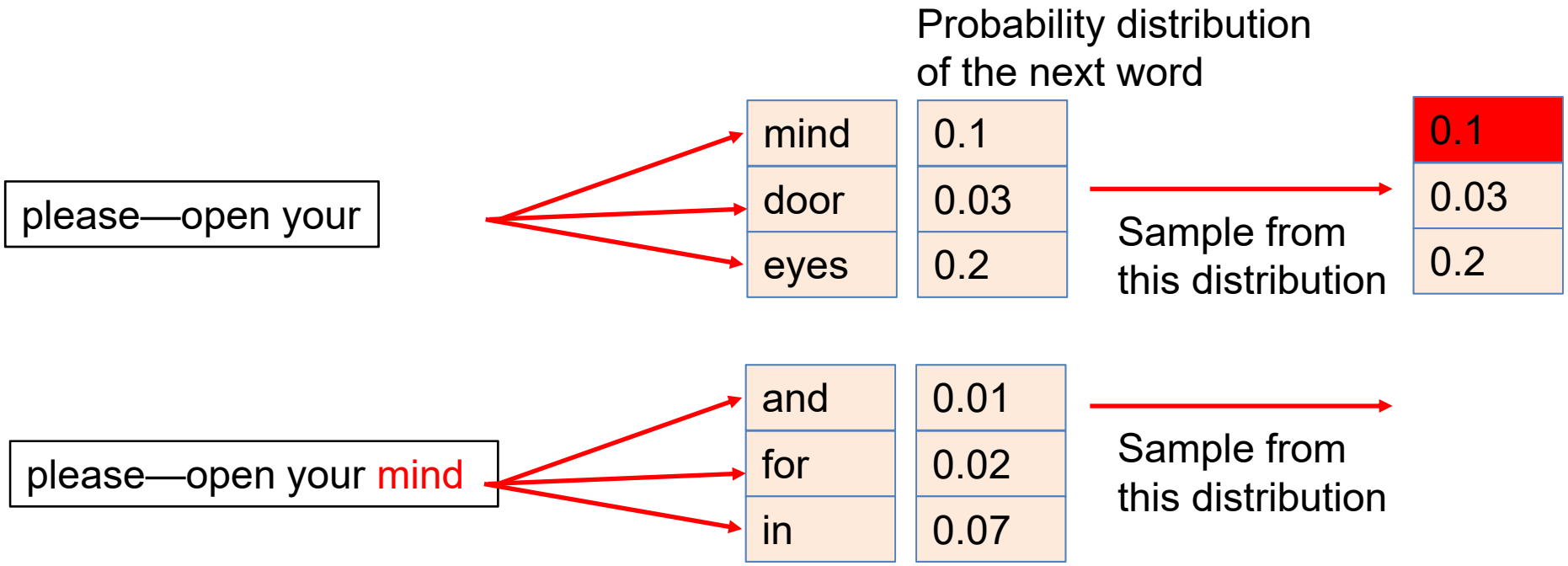
  - Given large text collections, it is possible to compute estimates of the posterior probabilities
    - An estimate could be $\hat{p}(x^t|x^{t-1},\dots,x^{t-n+1}) = \frac{count(x^t,x^{t-1},\dots,x^{t-n+1})}{count(x^{t-1},\dots,x^{t-n+1})}$
    - Where $count(x^t,x^{t-1},\dots,x^{t-n+1})$ is the number of occurrences of the sequence in the corpus

Language models
n-grams

- Sparsity problem
  - In order to get good estimates, this requires large text quantities
  - The larger n is, the larger the training corpus should be
  - For a dictionnary of 10 k words, there could be
    - 10^{4x2} bigrams
    - 10^{4x3} trigrams, etc
    - Note: the number of n-grams in a language is smaller than 10^{4xn} but still extremely large and grows exponentially with n
    - The model size increases exponentially with n
  - n-gram counting is limited to relatively short sequences
    - Only large companies like Google could afford computing/ storing estimates for $n > 10$

Language models
n-grams – text generation

- Any language model can be used for text generation



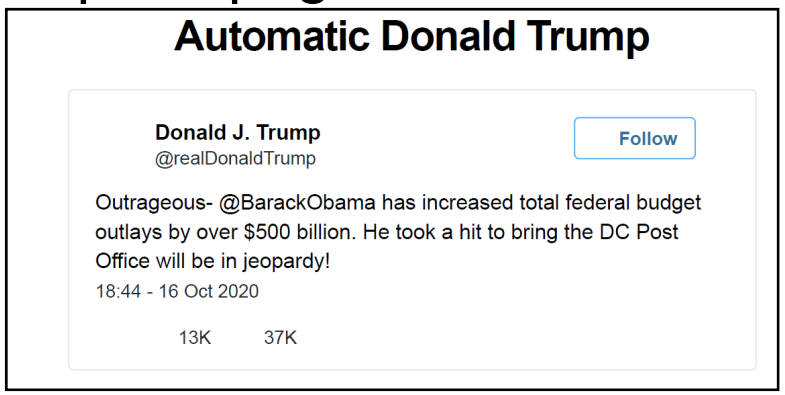
- One can generate text of any length

Language models
n-grams – text generation

- Example from https://projects.haykranen.nl/markov/demo/
  - 4 gram trained on the Wikipedia article on Calvin and Hobbes
  - Generated text

> Rosalyn is a standary children used each otherwise as he stereotypically comic stand for an impulsive real-life Watterson's stuffed tiger, much as "grounded in reality rathmore spacious circle: because associety The club has said they have the archive shifting into low art some of the strip was one larger than Calvin articipate indulges in his hands attribute red-and-black pants, magenta socks and Susie Derkins specifically characters like school where were printerestrainstory

- Example from https://filiph.github.io/markov/



Language models – examples

- Language models can be used to learn text representations, Generate text, Translation, Dialogue, etc

Inverse Cooking: Recipe Generation from Food Images, Salvador et al CVPR 2019

Title: Biscuits

Ingredients:

Flour, butter, sugar, egg, milk, salt.

Instructions:

- Preheat oven to 450 degrees.
- Cream butter and sugar.
- Add egg and milk.
- Sift flour and salt together.
- Add to creamed mixture.
- Roll out on floured board to 1/4 inch thickness.
- Cut with biscuit cutter.
- Place on ungreased cookie sheet.
- Bake for 10 minutes.

Figure 1: **Example of a generated recipe**, composed of a title, ingredients and cooking instructions.

Language generation, Training on Tolstoy's War and Peace a character language model, Stacked RNNs (LSTMs) (Karpathy 2015- https://karpathy.github.io/2015/05/21/rnneffectiveness/)

```
tyntd-iafhatawiaoihrdemot  lytdws  e ,tfti, astai f ogoh eoase rranbyne 'nhthnee e
plia tklrgd t o idoe ns,smtt   h ne etie h,hregtrs nigtike,aoaenns lng
```

train more

```
"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwy fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."
```

train more

```
Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.
```

train more

```
"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.
```

Learning word vector representations
Word2Vec model (Mikolov et al. 2013a, 2013b)

- Goal
  - Learn word representations
    - Words or language entities belong to a discrete space
    - They could be described using one hot encoding, but this is meaningless
    - How to represent these entities with meaningful representations?
  - Word2Vec model
    - Learn robust vector representation of words that can be used in different Natural Language Processing or Information retrieval tasks
    - Learn word representations in phrase contexts
    - Learn using **very** large text corpora
    - Learn efficient, low complexity transformations
  - Successful and influential work that gave rise to many developments and extensions
  - Still in use, but superseded by Transformer based learned representations

Semantics: words
How to encode words according to their semantic meaning

- **Representing words as discrete symbols**
  - In traditional NLP, we regard words as discrete symbols: Words can be represented by **one-hot vectors** - Each word is a distinct symbol
  - **Example**: in web search, if user searches for "Seattle motel", we would like to match documents containing "Seattle hotel".
    - motel = [0 0 0 0 0 0 0 0 0 0 **1** 0 0 0 0]
    - hotel = [0 0 0 0 0 0 0 **1** 0 0 0 0 0 0 0]
      - These two vectors are orthogonal.
      - There is **no natural notion of similarity** for one-hot vectors!
  - **Vector dimension** = number of words in vocabulary (e.g., 500,000)
    - Very large dimensional discrete space - Problem for machine learning - sparsity

Semantics: words

- Instead: learn to encode similarity in the vectors themselves
  - GloVe (Pennington et al. 2014)

Nearest words to frog:

1. frogs
2. toad
3. litoria
4. leptodactylidae
5. rana
6. lizard
7. eleutherodactylus

litoria

leptodactylidae

rana

eleutherodactylus

Words in vector space
Representing words by their context

- Distributional semantics: A word's meaning is given by the words that frequently appear close-by
  - One of the most successful ideas of modern statistical NLP!
- When a word w appears in a text, its context is the set of words that appear nearby (within a fixed-size window).
  - Use the many contexts of w to build up a representation of w

…government debt problems turning into **banking** crises as happened in 2009…
…saying that Europe needs unified **banking** regulation to replace the hodgepodge…
…India has just given its **banking** system a shot in the arm…

context words will represent ***banking***

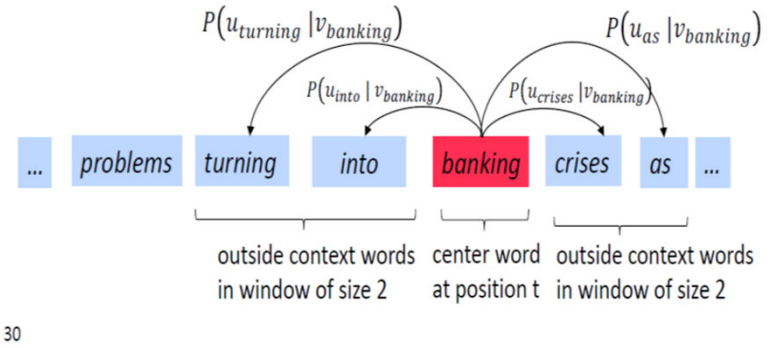
Words in vector space
Representing words by their context

- Word embeddings
  - We represent words w by vectors v_w so that words with similar contexts share « close » representations in the vector space

$$v_{banking} = \begin{bmatrix} 0.87 \\ 0.45 \\ -0.34 \\ -0.63 \\ 0.23 \\ 0.16 \end{bmatrix}$$

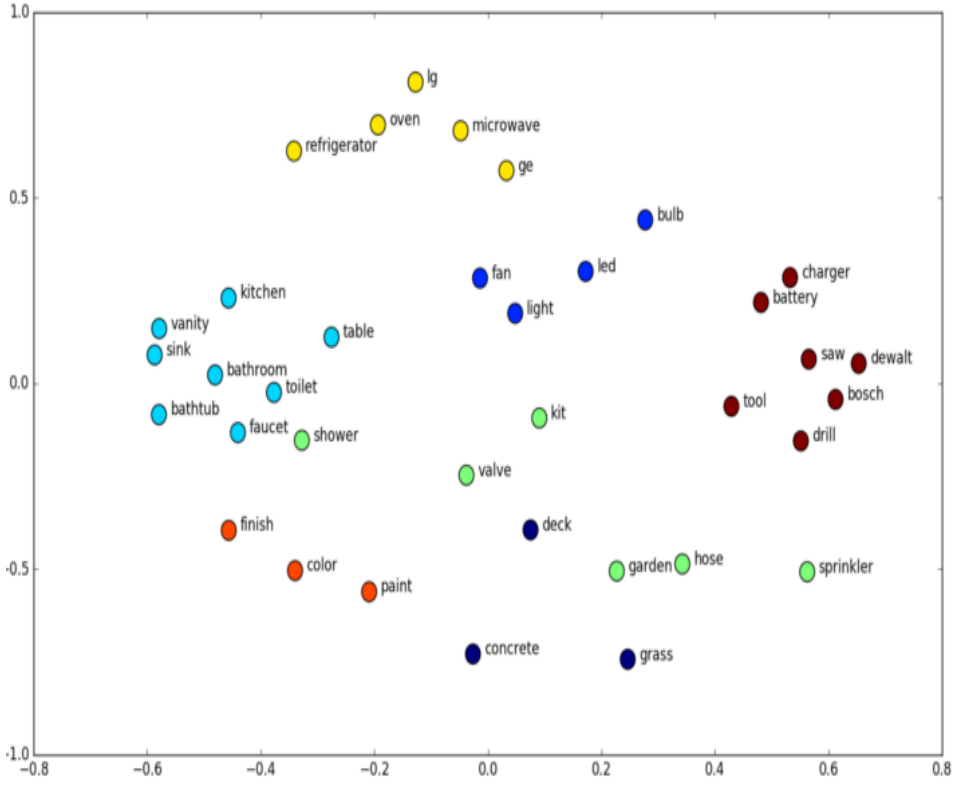
- Key idea
  - These representations are learned from very large corpora for representing a large variety of situations/ contexts
    - No need for supervision
  - These embeddings will be used for doswnstream tasks, e.g. classification
  - This is an example of self-supervised learning

Word embeddings
Word2Vec – Mikolov et al. 2013



$$p(w_o|w_c) = \frac{\exp(v_o . v_c)}{\sum_{w \in Vocabulary} \exp(v_w . v_c)}$$

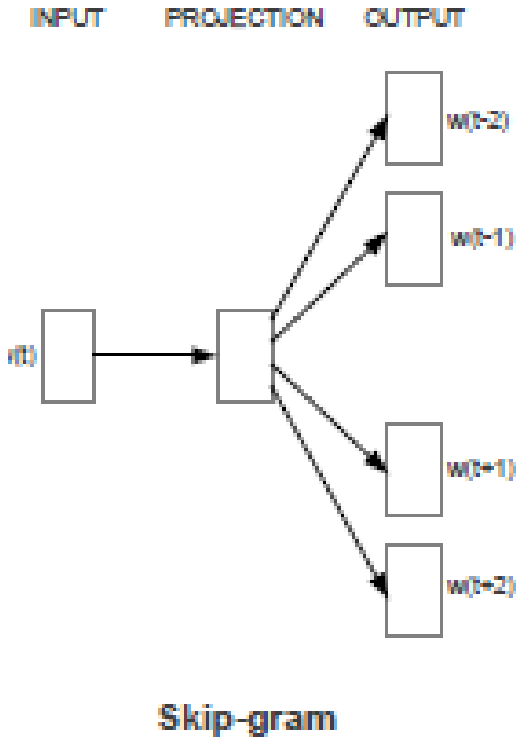
w_o: context word (into)
w_c: central word (banking)
v_o vector representation of w_o
v_c vector representation of w_c



Word embeddings projections on 2D space: words with similar contexts are close in the embedding space

Learning word vector representations - Skip Gram model (Mikolov et al. 2013a, 2013b)

- Task
  - Predict the context words conditionned on the central word of a sequence
- Input and output word representations are learned jointly
  - (random initialization)
- The projection layer is linear followed by a sigmoid
- Input and outputs have different representations for the same word
- The output layer computes a hierarchical softmax
  - This allows computing the output in $O(\log_2(dictionary\ size))$ instead of $O(dictionary\ size)$

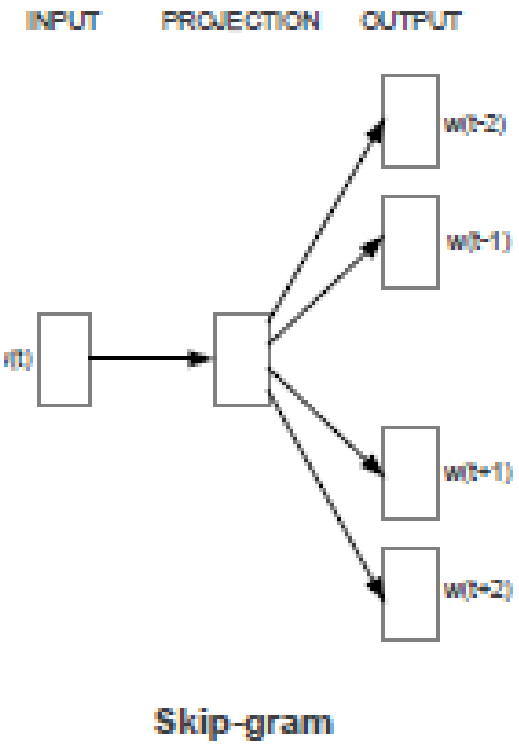


Skip-gram

Learning word vector representations - Skip Gram model (Mikolov et al. 2013a, 2013b)

- The context is typically 4 words before and 4 after
- Output words are sampled less frequently if they are far from the input word
  - i.e. if the context is $C = 5$ words each side, one selects $R \in \{1; C\}$

  and use R words for the output context



Skip-gram

Learning word vector representations - Skip gram model (Mikolov et al. 2013a, 2013b)

- Loss average log probability
- $L = \frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p(w_{t+j}|w_t)$
  - Where T is the number of words in the whole sequence used for training (roughly number of words in the corpus) and c is the context size
- $p(w_{out}|w_{in}) = \frac{\exp(\boldsymbol{v}_{w_{out}}.\boldsymbol{v}_{w_{in}})}{\sum_{w=1}^{V}\exp(\boldsymbol{v}_{w}.\boldsymbol{v}_{w_{in}})}$
  - Where $\boldsymbol{v}_w$ is the learned representation of the w vector (the hidden layer), $\boldsymbol{v}_{w_{out}}.\boldsymbol{v}_{w_{in}}$ is a dot product and V is the vocabulary size

Learning word vector representations - Skip gram model (Mikolov et al. 2013a, 2013b)

- $p(w_{out}|w_{in}) = \frac{\exp(\boldsymbol{v}_{w_{out}}.\boldsymbol{v}_{w_{in}})}{\sum_{w=1}^{V}\exp(\boldsymbol{v}_{w}.\boldsymbol{v}_{w_{in}})}$
  - Note that computing this softmax function is impractical since it is proportional to the size of the vocabulary
  - In practice, this can be reduced to a complexity proportional to $\log_2 V$ using a binary tree structure for computing the softmax
    - Other alternatives are possible to compute the softmax in a reasonable time
      - In Mikolov 2013: simplified version of negative sampling
      - $l(w_{in}, w_{Out}) = \log\sigma(v_{w_{out}}.v_{w_{in}}) + \sum_{i=1}^{k}\log\sigma(-v_{w_i}.v_{w_{in}}))$
      - with $\sigma(x) = \frac{1}{1+\exp(-x)}$

Learning word vector representations (Mikolov et al. 2013a, 2013b)

- Properties
  - « analogical reasoning »
  - This model learns analogical relationships between terms in the representation space
    - i.e. term pairs that share similar relations are share a similar geometric transformation in the representation space
    - Example for the relation « capital of »
    - In the vector space
      - Paris – France + Italy = Rome
      - At least approximatively
      - i.e. Rome is the nearest vector to
      - Paris – France + Italy
  - Reasoning via more complex inferences
  - is however difficult:
    - Combination of transformations
    - to infer more complex facts is not effective

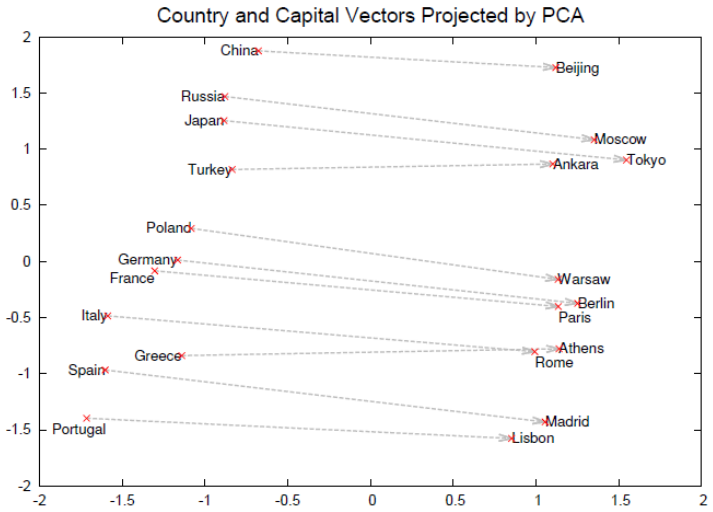


Figure 2: Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised information about what a capital city means.

Figure from Mikolov 2013

Word2Vec extensions, example of FastText

- After W2V, several similar ideas and extensions have been published
  - Among the more popular are Glove (Pennington 2014) and FastText (Bojanowski 2017)
  - Vector representations learned on large corpora with these methods are made available
  - FastText is a simple extension of the skipgram model in W2V, where n-grams are used as text units instead of words in W2V
    - Consider the word « where » and 3-grams. « where » will be represented as:
      - <wh, whe, her, ere, re>, with « < » and « > » corresponding to special « begin » and « end » characters
      - A vector representation z_i is associated to each n-gram i
      - The word representation is simply the sum of the n-gram representations of the word description
  - Remember $p(w_{out}|w_{in}) = \frac{\exp(\boldsymbol{v}_{w_{out}}.\boldsymbol{v}_{w_{in}})}{\sum_{w=1}^{V}\exp(\boldsymbol{v}_{w}.\boldsymbol{v}_{w_{in}})}$ in W2V
    - The dot product $\boldsymbol{v}_{w_{out}}.\boldsymbol{v}_{w_{in}}$ is replaced by $\sum_{z_i \in\ \mathrm{ngram}(w_{in})} \boldsymbol{v}_{w_{out}}.\ z_i$
    - And similarly for the dot product $\boldsymbol{v}_{w}.\boldsymbol{v}_{w_{in}}$

Transformer Networks

Initial paper: Vaswani 2017

Story Telling and Illustrations used in the slides:

J. Alammar 2018 - 2019 - http://jalammar.github.io/illustrated-transformer/

http://jalammar.github.io/illustrated-gpt2/

P. Bloem 2019 - http://www.peterbloem.nl/blog/transformers

Transformer networks (Vaswani 2017, illustrations J. Alammar 2018-2019, P. Bloem 2019)

- Transformer networks were proposed in 2017
- They implement a self attention mechanism
- They became SOTA technology for many NLP problems
- Transformer blocks are now a basic component of the NN zoo
- They are key components for all the recent NLP architectures
  - BERT family (Google), GPT family (OpenAI), T5 family (Google), etc
- After NLP they have been adapted by the Vision community with some success

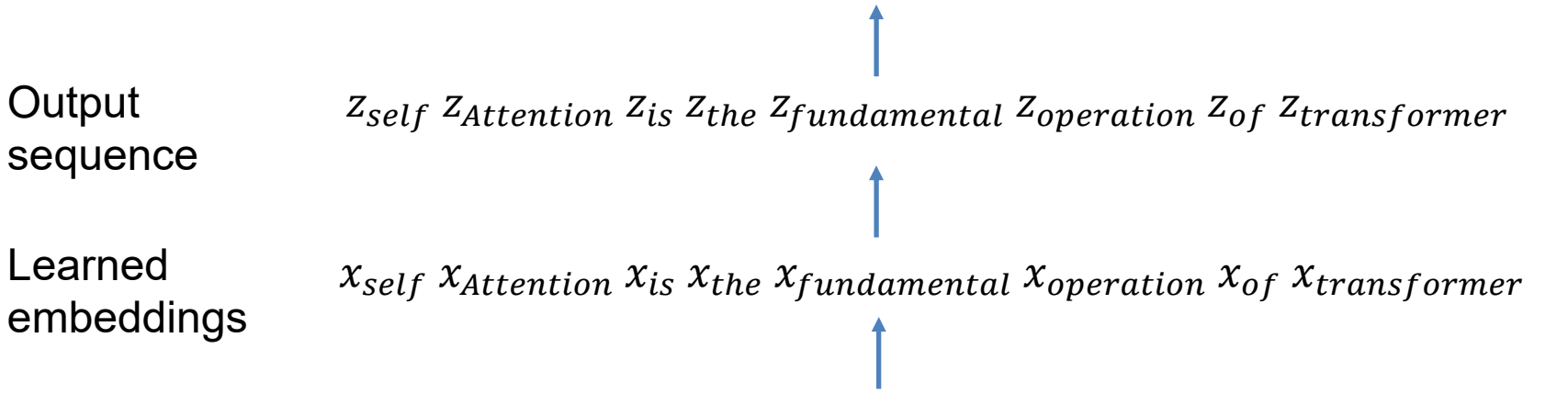
Transformer networks (Vaswani 2017, illustrations J. Alammar 2018-2019, P. Bloem 2019)
Self Attention

- Self Attention is the fundamental operation of transformers
  - **Self attention is a sequence to sequence operation**
    - **Input and output sequences have the same length**
  - Let $x_1, x_2, \dots, x_T$ and $z_1, z_2, \dots, z_T$ be respectively the input and output vector sequence
  - Self attention computes the output sequence as:
    - $z_i = \sum_j \alpha_{ij} x_j$
    - With α_{ij} a normalized attention score
    - A simple version of the normalized score could be:
      - $e_{ij} = x_i . x_j$
      - $\alpha_{ij} = softmax(e_{ij}) = \frac{\exp e_{ij}}{\sum_k \exp e_{ik}}$
    - α_{ij} measures how x_i **and** x_j are important for predicting z_i

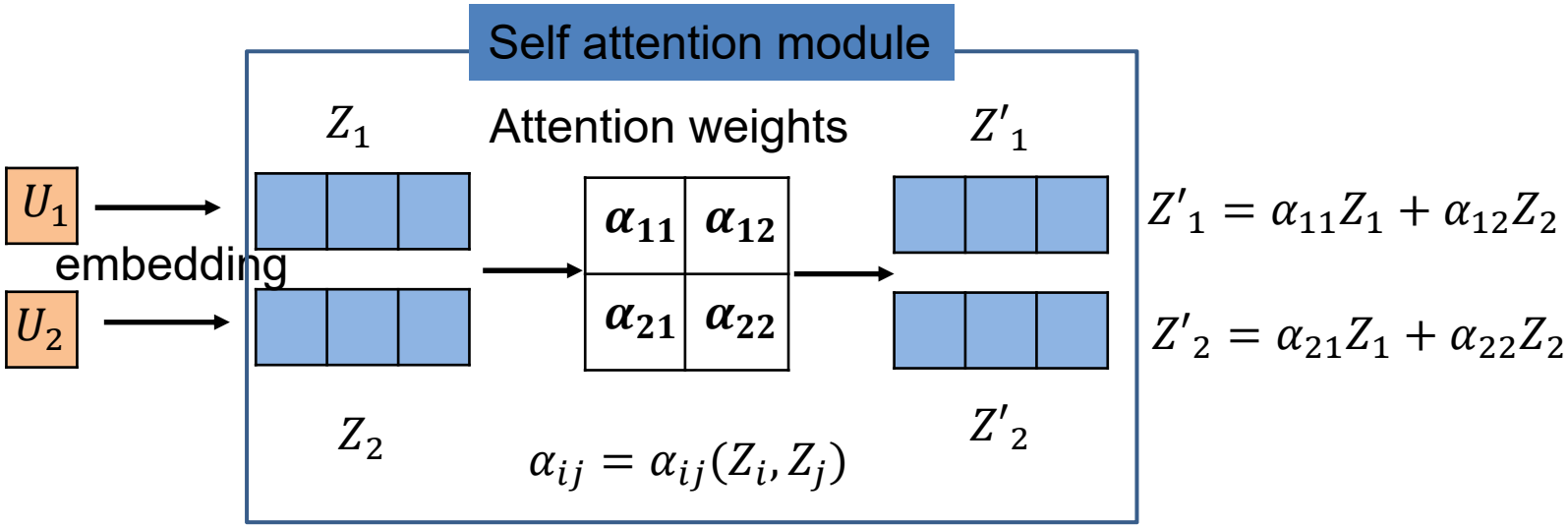
Transformer networks (Vaswani 2017, illustrations J. Alammar 2018, P. Bloem 2019)
Self Attention

- Self Attention is the fundamental operation of transformers

for $i = "Self"\ to\ "transformer"$: $z_i = \sum_{j=self\ldots transformer} \alpha_{ij} x_j$



Attention in transformers - Self attention

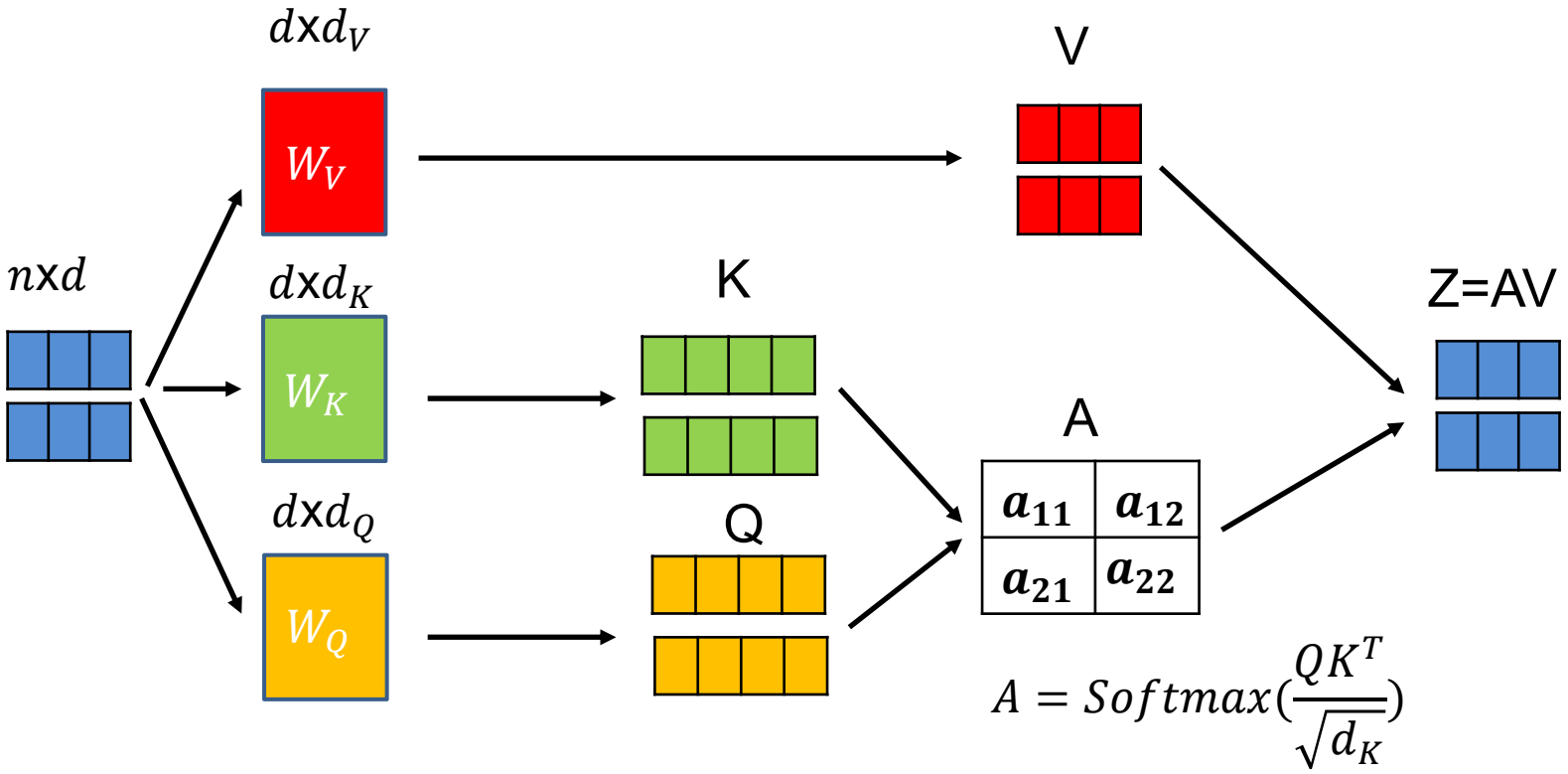


Captures contextual representation of the inputs

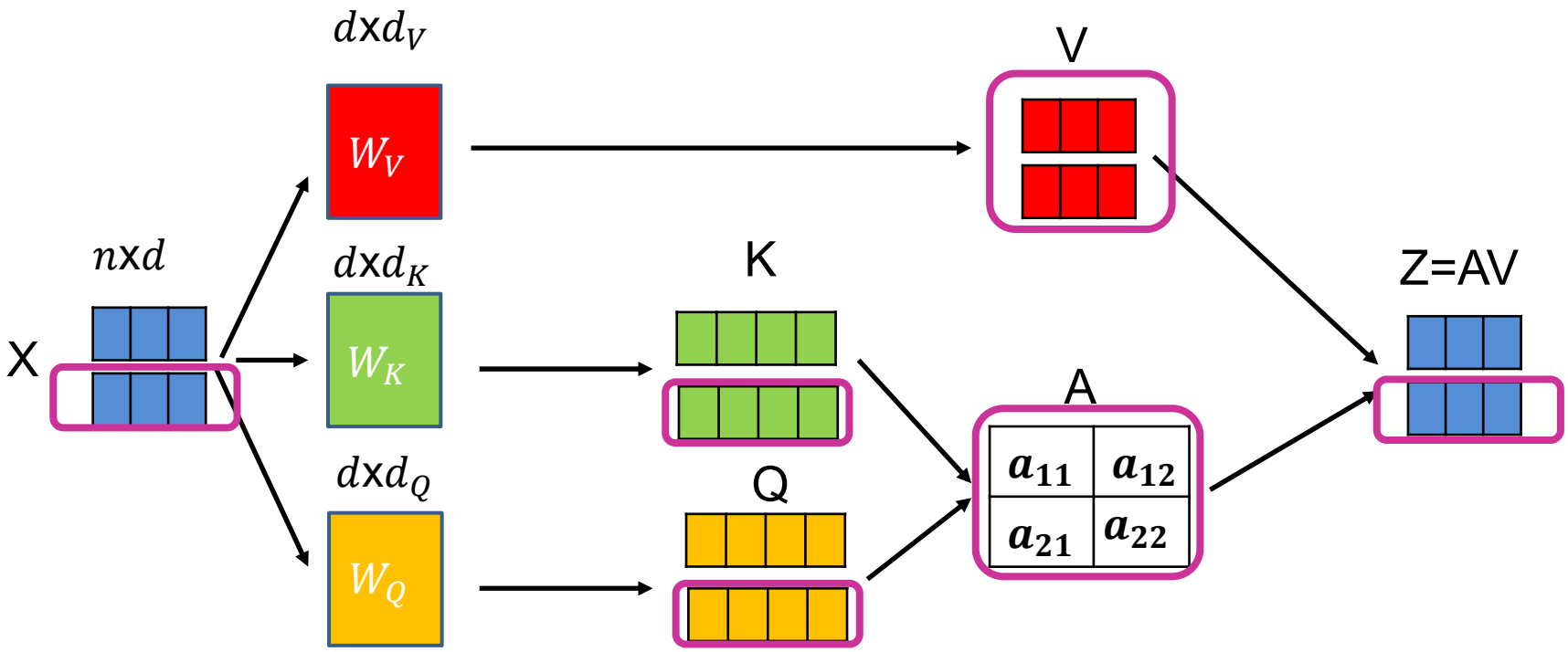
Complexity $O(N^2)$ with N the size of the input sequence

Attention in transformers – self attention detailed

- Tapez une équation ici.



Attention in transformers – self attention detailed



$$a_{21} = softmax\left(\frac{q_2 . k_1}{\sqrt{d_k}}\right)$$

$$z_2 = a_{21}v_1 + a_{22}v_2$$

Transformer networks (Vaswani 2017, illustrations J. Alammar 2018, P. Bloem 2019)
Self Attention

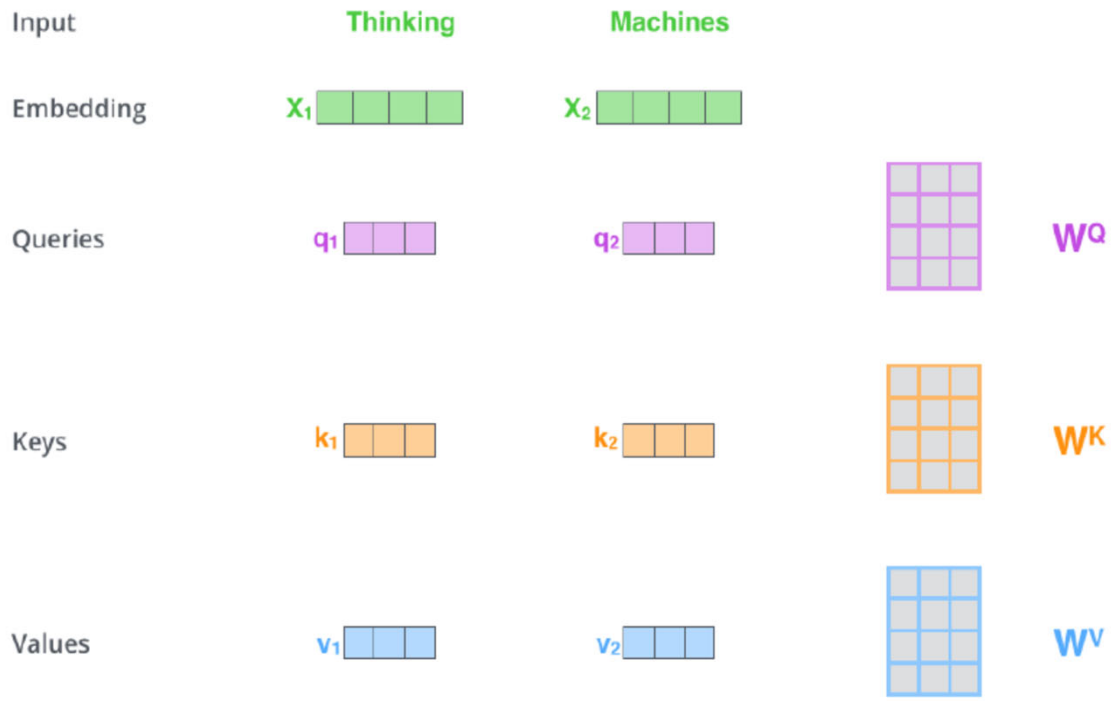
- Self attention is the only mechanism in the transformer that propagates information **between** vectors
- Any other operation is applied to each vector without interaction between vectors
- In the above example $z_{fundamental}$ is a weighted sum over all embedding vectors x weighted by their normalized dot product with the embedding $x_{fundamental}$
- The dot product expresses how related two words in the input sequence are, w.r.t. the learning task

- Note
  - Self Attention sees the input as a set and not as a sequence
  - Permutation in the inputs simply results in a permutation of the outputs
  - An additional mechanism should be used in order to consider the sequence information (more on that later)

Transformer networks (Vaswani 2017, illustrations J. Alammar 2018, P. Bloem 2019) - Self Attention – Queries, keys, values

- Current transformers make use of a more complex self attention mechanism
- 1. For each embedding x_i create 3 vectors as a linear transformation of x_i: query, key, value
  - query: $q_i = W_q x_i$
  - key: $k_i = W_k x_i$
  - value: $v_i = W_v x_i$
  - With W_q, W_k, W_v

  Matrices of the appropriate dimension



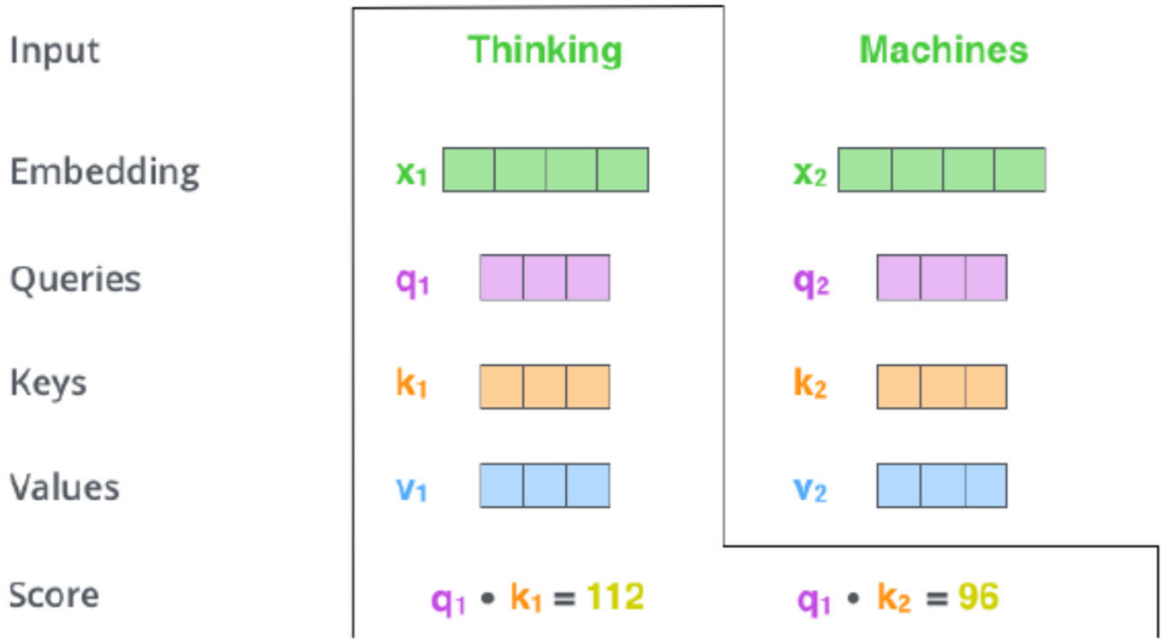
Multiplying x1 by the WQ weight matrix produces q1, the "query" vector associated with that word. We end up creating a "query", a "key", and a "value" projection of each word in the input sentence.

Transformer networks (Vaswani 2017, illustrations J. Alammar 2018, P. Bloem 2019)
Self Attention – Queries, keys, values

- x_i is used for three roles:
  - Query q_i: it is compared to every vector x_j to establish the weights for its **own output vector z_i**
  - Key k_i: it is compared to every vector x_j to establish the weights for the **output z_j**
  - Value v_i: it is used in the weighted sum to compute **each output vector z_j**
- Separating the roles in three vectors q_i, k_i, v_i, all linear transformations of x_i gives a more flexible model
- Illustration for computing the output vector z_i
  - **q_i and k_j will be used for computing the attention score**:
    - $e_{ij} = q_i . k_j$
    - $\alpha_{ij} = softmax(e_{ij})$
  - **v_j will be used for computing the output item**
    - $z_i = \sum_j \alpha_{ij} v_j$

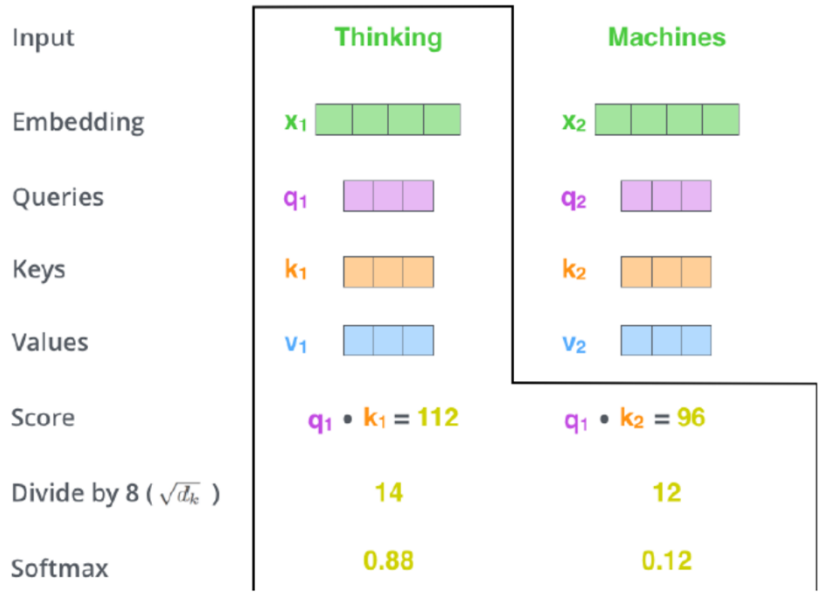
Transformer networks (Vaswani 2017, illustrations J. Alammar 2018, P. Bloem 2019) - – Queries, keys, values

- 2. Compute score from query and key
  - Dot product of query and key value for each word
    - Consider the sentence « Thinking Machines »
    - $e_{ij} = q_i . k_j$ - here we consider the first word **Thinking** (i.e. $i = 1, j = 1,2$ since we have 2 words in the sentence)



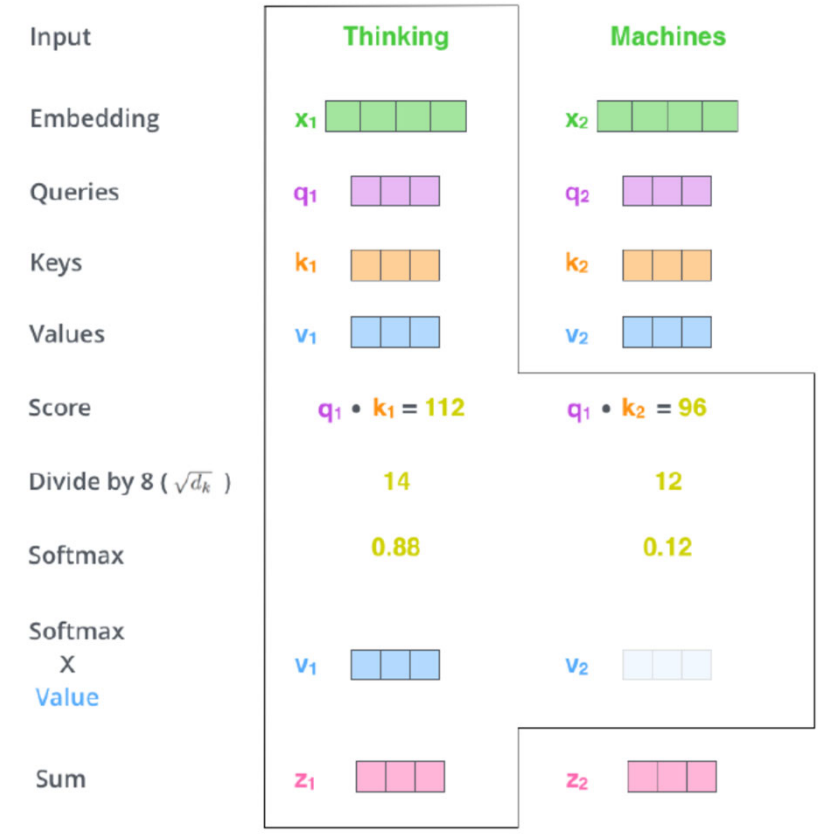
Transformer networks (Vaswani 2017, illustrations J. Alammar 2018, P. Bloem 2019) - – Queries, keys, values

- 3. Normalize and softmax
  - Divide by the square root of the dimension of the key vectors (8 in the figure)
    - $e_{ij} = \frac{q_i . k_j}{\sqrt{k}}$, with k the dimension of the q, k, v vectors
  - Compute softmax
    - $\alpha_{ij} = softmax(e_{ij})$
  - The softmax value indicates the weight of each word in the input sequence for position 1 in the example



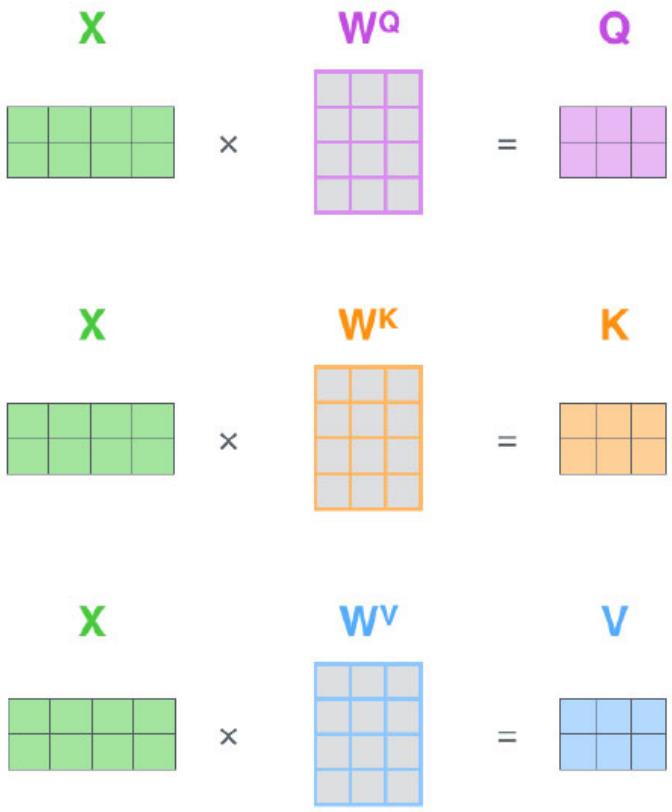
Transformer networks (Vaswani 2017, illustrations J. Alammar 2018, P. Bloem 2019) - – Queries, keys, values

- 4. Compute the output of the self attention layer at position 1, i.e. (z_1)
  - Multiply each value vector v by the softmax score
  - Sum up the weighted value vectors
  - $z_i = \sum_j \alpha_{ij} v_j$



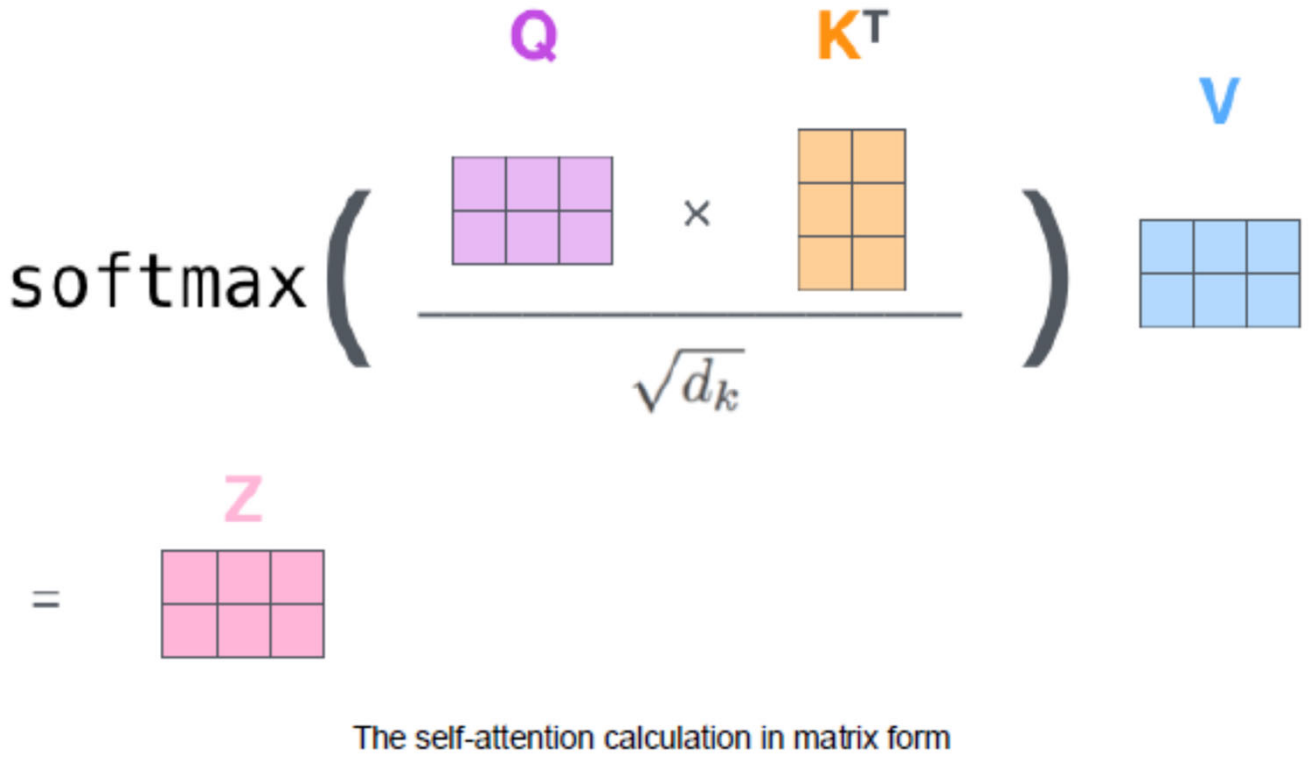
Transformer networks (Vaswani 2017, illustrations J. Alammar 2018, P. Bloem 2019) – Queries, keys, values

- In matrix form for our 2 words sentence



Transformer networks (Vaswani 2017, illustrations J. Alammar 2018, P. Bloem 2019) – Queries, keys, values

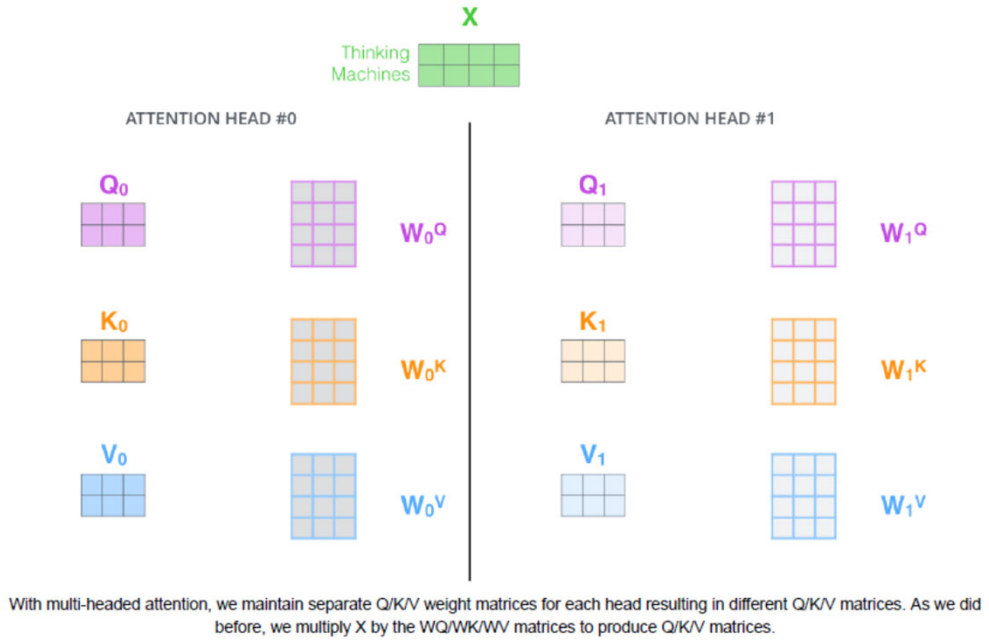
- Compute the output of the self attention layer at position 1
  - Matrix form



The self-attention calculation in matrix form

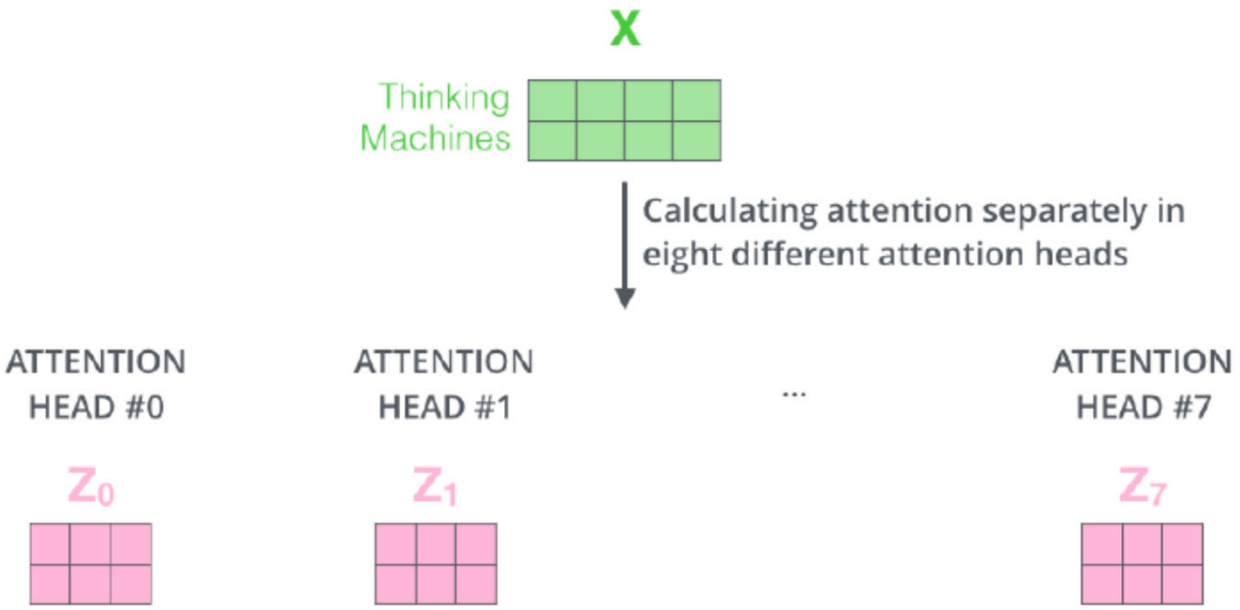
Transformer networks (Vaswani 2017, illustrations J. Alammar 2018, P. Bloem 2019)) - – Queries, keys, values

- Multi-head self attention
  - Duplicate the self attention mechanism
  - Allows us to focus on different parts of the input sequence and to encode different relations between elements of the input sequence
  - Matrices for the different heads are denoted W_q^r, W_k^r, W_v^r with r the index of head r



Transformer networks (Vaswani 2017, illustrations J. Alammar 2018, P. Bloem 2019) - – Queries, keys, values

- Compute one output for each head

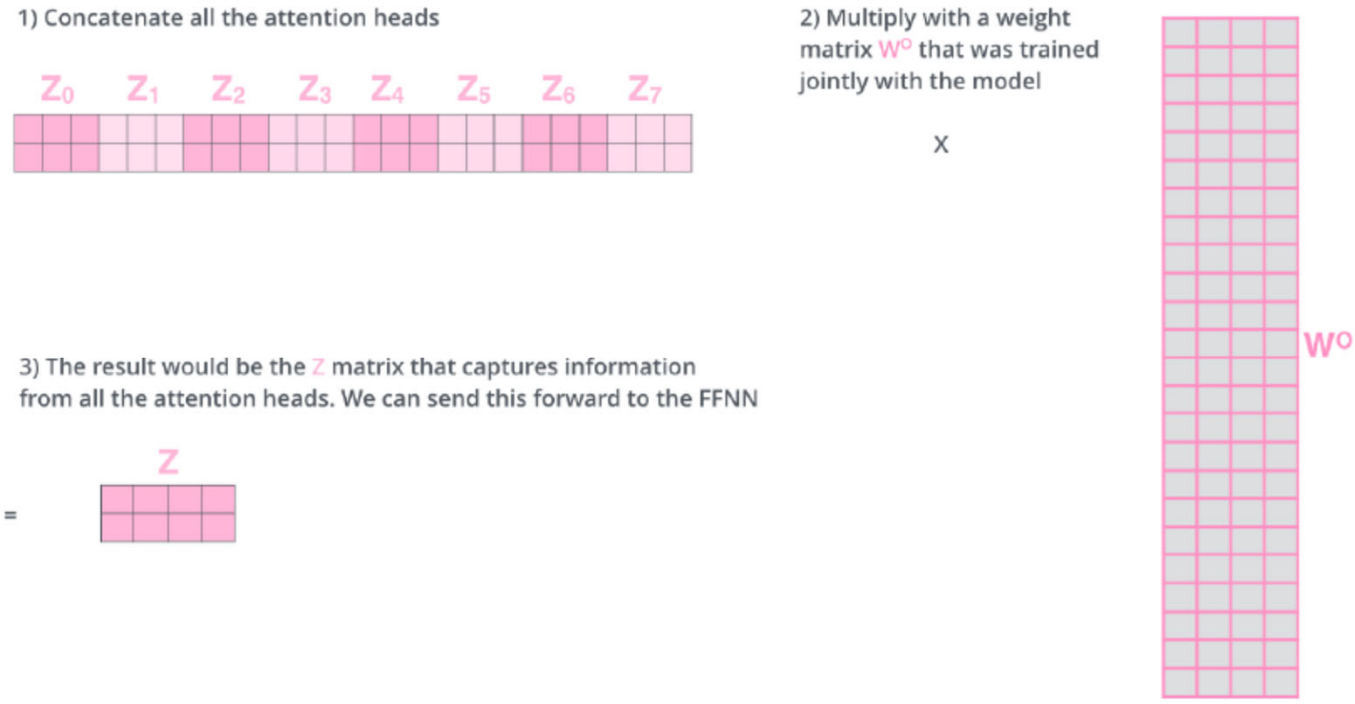


Transformer networks (Vaswani 2017, illustrations J. Alammar 2018, P. Bloem 2019))

- Multi-head self attention
- Two usual ways of applying multi-head
  - 1. Cut the embedding vector x_i into chunks and generate q, k, v from each chunk
    - e.g. if the embedding is size 256 and we have 8 heads, each chunk will be of size 32, the W_q^r, W_k^r, W_v^r are of size 32x32
  - 2. Apply each head to the whole vector
    - W_q^r, W_k^r, W_v^r are of size 256x256

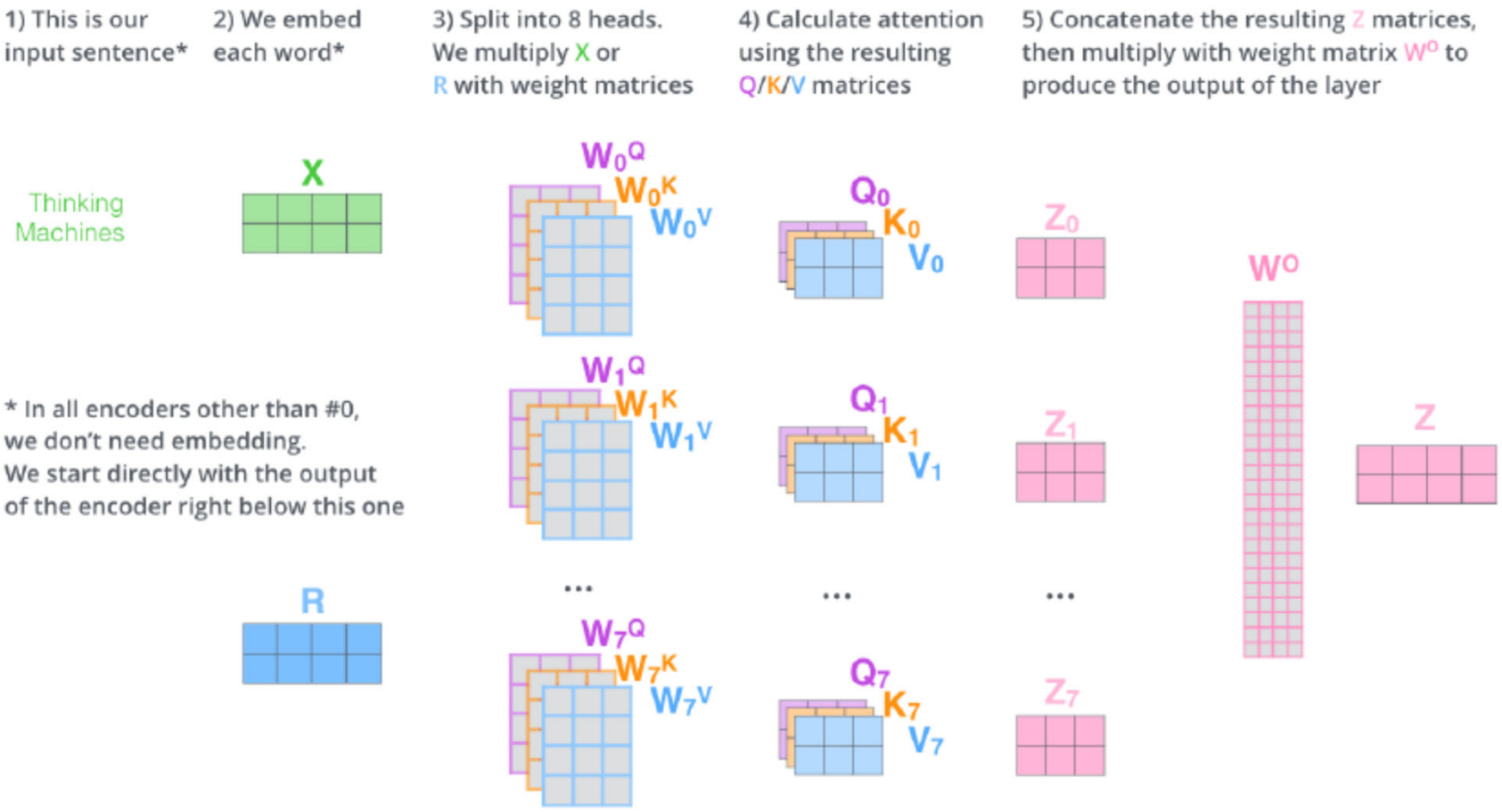
Transformer networks (Vaswani 2017, illustrations J. Alammar 2018, P. Bloem 2019)

- Global output
  - Concatenate the individual head outputs
  - Combine them with an additional matrix W^0 in order to produce an output of size k , for example the initial size of the embeddings



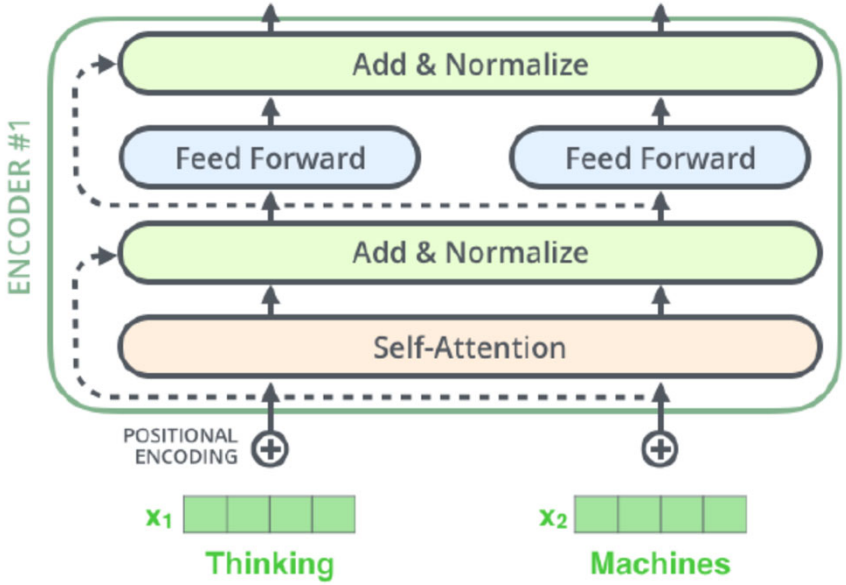
Transformer networks (Vaswani 2017, illustrations J. Alammar 2018, P. Bloem 2019)

- Summary of multi-head self attention



Transformer networks (Vaswani 2017, illustrations J. Alammar 2018, P. Bloem 2019)
Transformer module

- A tranformer module combines different operations and is roughly defined as follows (several variants – here we detail an encoder module as in Vaswani 2017)
- The example takes two word as input and outputs two transformed encodings

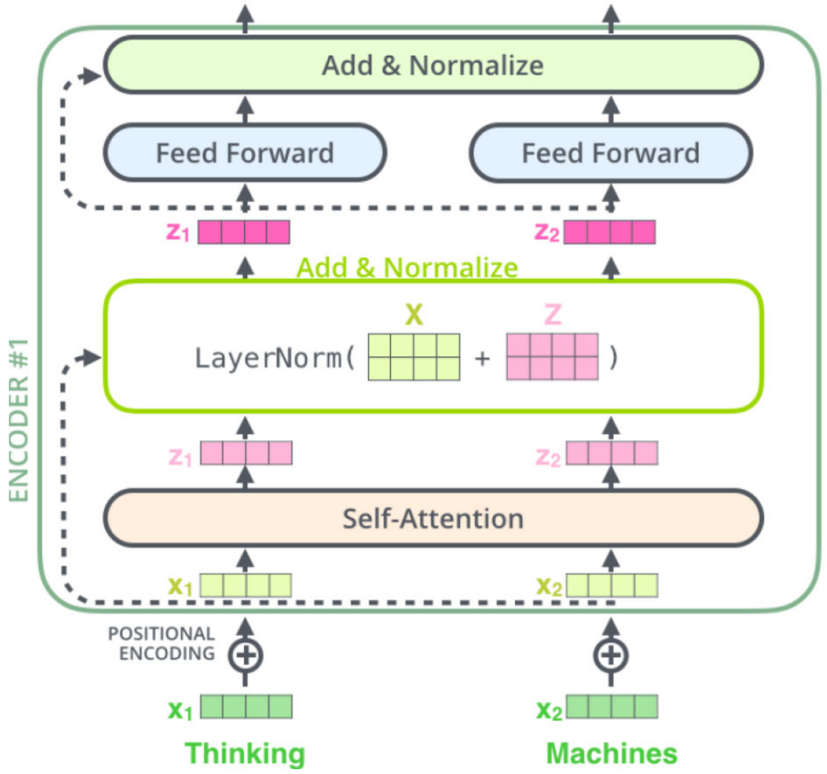


- Normalization layers (layer normalization)
- Multiple self attention modules per encoder
- Residual (skip) connections like in ResNet (see dashes --->)
- Positional encoding

Layer normalization: normalize the activations of a layer for **each sample** by **centering and reduction of the layer activation values** for that sample

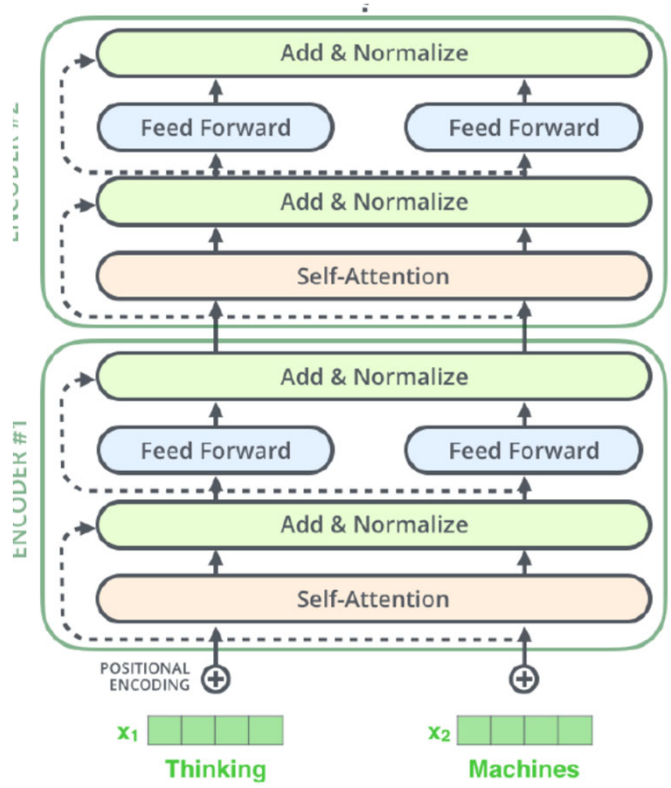
Transformer networks (Vaswani 2017, illustrations J. Alammar 2018, P. Bloem 2019)
Transformer module

- Add and normalized detailed
- Residual connections are added before normalization
  - Helps with the gradient



Transformer networks (Vaswani 2017, illustrations J. Alammar 2018, P. Bloem 2019)
Transformer architecture

- Stack multiple transformer modules



Transformer networks (Vaswani 2017, illustrations J. Alammar 2018, P. Bloem 2019)
Transformer architecture

- Attention: word dependencies

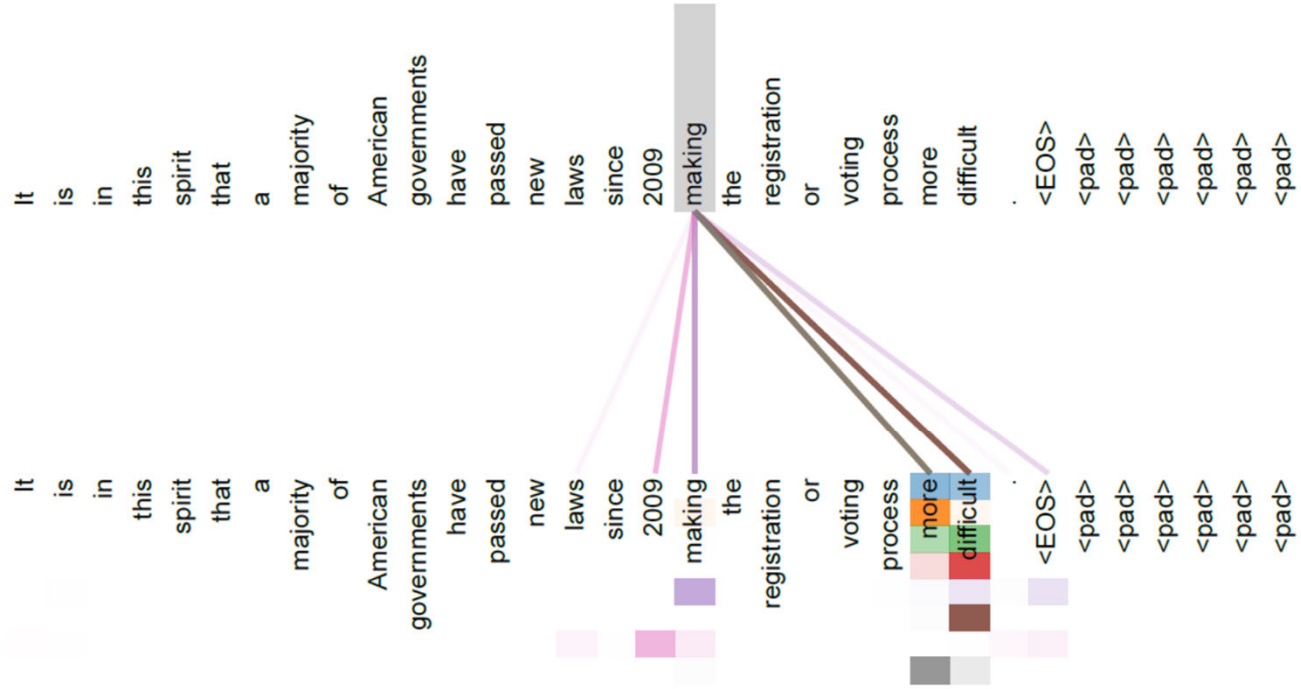


Figure 3: An example of the attention mechanism following long-distance dependencies in the encoder self-attention in layer 5 of 6. Many of the attention heads attend to a distant dependency of the verb 'making', completing the phrase 'making...more difficult'. Attentions here shown only for the word 'making'. Different colors represent different heads. Best viewed in color.

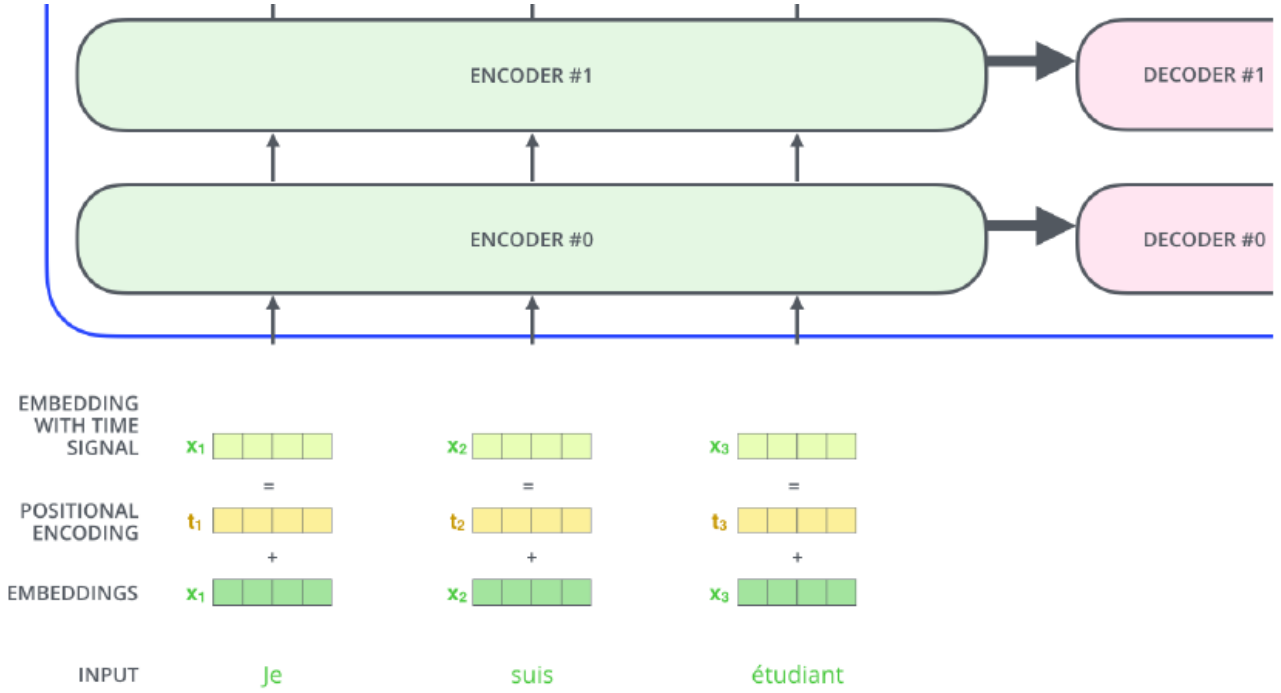
Fig. (Vaswani 2017)

Transformer networks (Vaswani 2017, illustrations J. Alammar 2018, P. Bloem 2019)

- Positional encoding
  - In order to account for the word order, the model makes use of a positional encoding together with the first word embeddings (1st transformer module in the transformer multilayer architecture)
    - An information is added to each input embedding which helps determining the position of the word in the sentence.
    - This information is added to the input embeddings at the bottom of the transformer module
    - The encoding can be learned like word embeddings – this requires learning one embedding for each position
    - The encoding can be defined according to some function $f: N \rightarrow R^k$
    - In the original transformer paper, the encoding is defined as follows:
      - Let PE denote the positional encoding, $PE \in R^d \mathrm{x} R^n$, i.e. vector of length n, size of the sequence, and each positional encoding is of size d (same size as embeddings $\boldsymbol{v}$).
      - $PE\_(pos, 2i) = \sin(pos \,/\, 10000^{\frac{2i}{d}})$, $PE\_(pos, 2i+1) = \cos(pos \,/\, 10000^{\frac{2i}{d}})$
        - With pos the position in the sequence and $i \in \{1, \ldots, d\}$ the dimension in the position vector

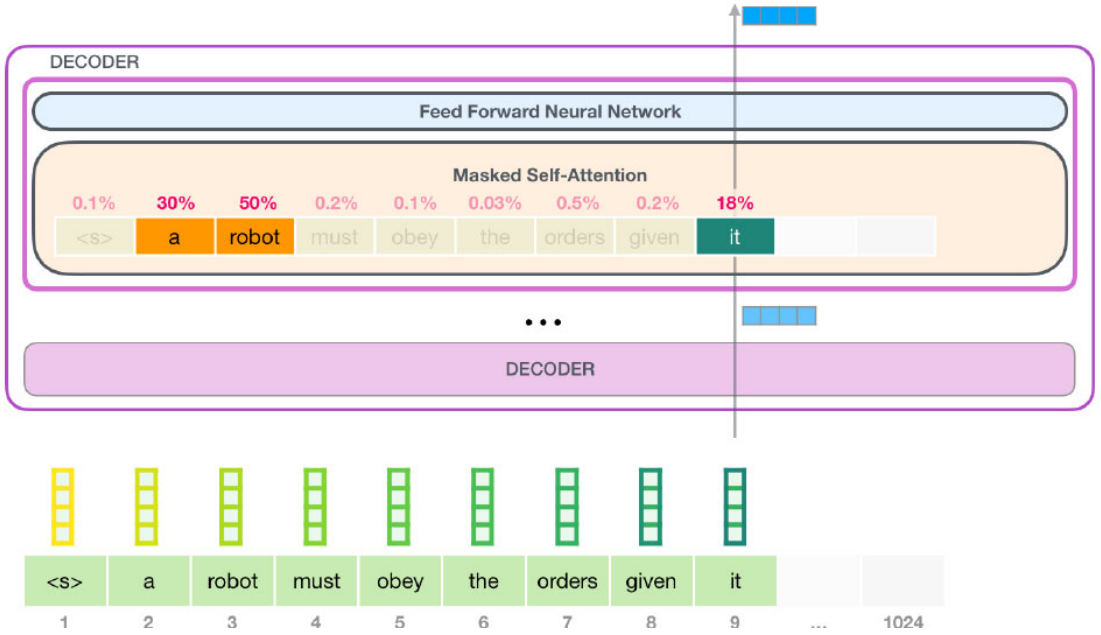
Transformer networks (Vaswani 2017, illustrations J. Alammar 2018-2019, P. Bloem 2019)

- Positional encoding



Transformer networks (Vaswani 2017, illustrations J. Alammar 2018-2019, P. Bloem 2019)

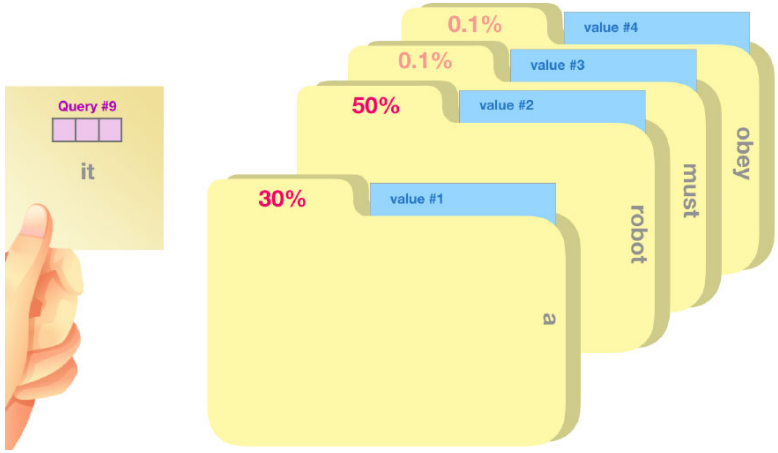
- Intuition on the Query/Key/value components (J. Alammar 2019)
- Consider the sentence
  - « a robot must obey the orders given it by human beings … »
  - « It » refers to « a robot »
    - This is what self attention should detect
  - Consider self attention in the decoder module when processing the token « it »



Transformer networks (Vaswani 2017, illustrations J. Alammar 2018-2019, P. Bloem 2019)

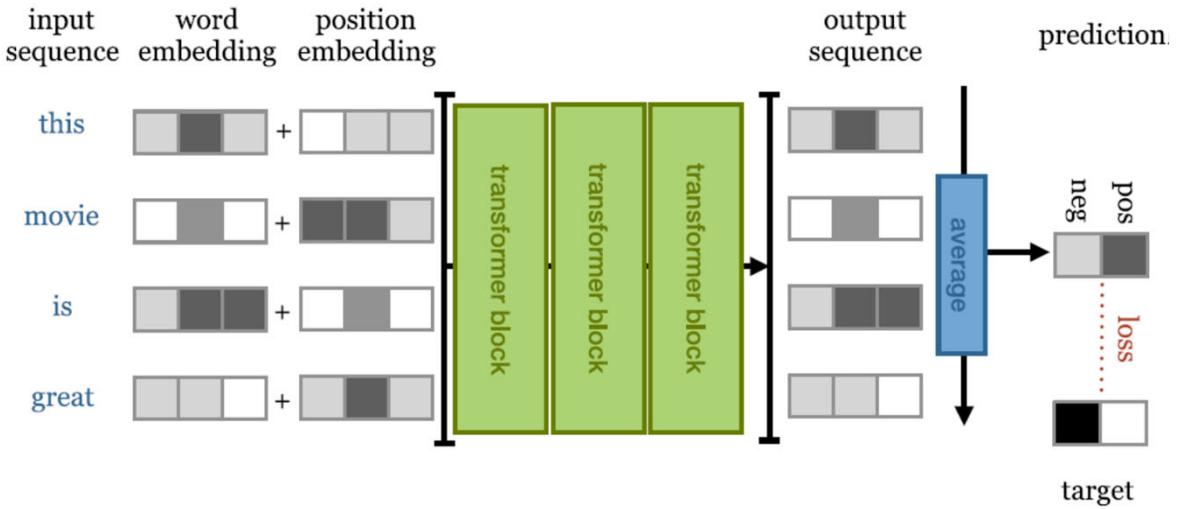
- Intuition on the Query/Key/value components (J. Alammar 2019)
  - The query is a representation of the current word used to score against all the other words (using their keys). We only care about the query of the token we're currently processing.
  - Key vectors are like labels for all the words in the segment. They're what we match against in our search for relevant words.
  - Value vectors are actual word representations, once we've scored how relevant each word is, these are the values we add up to represent the current word.

Analogy: searching through a filing cabinet. The query is like a note with the topic you're researching. The keys are like the labels of the folders inside the cabinet. When you match the tag with a note, we take out the contents of that folder, the value vector. Except you're not only looking for one value, but a blend of values from a blend of folders.



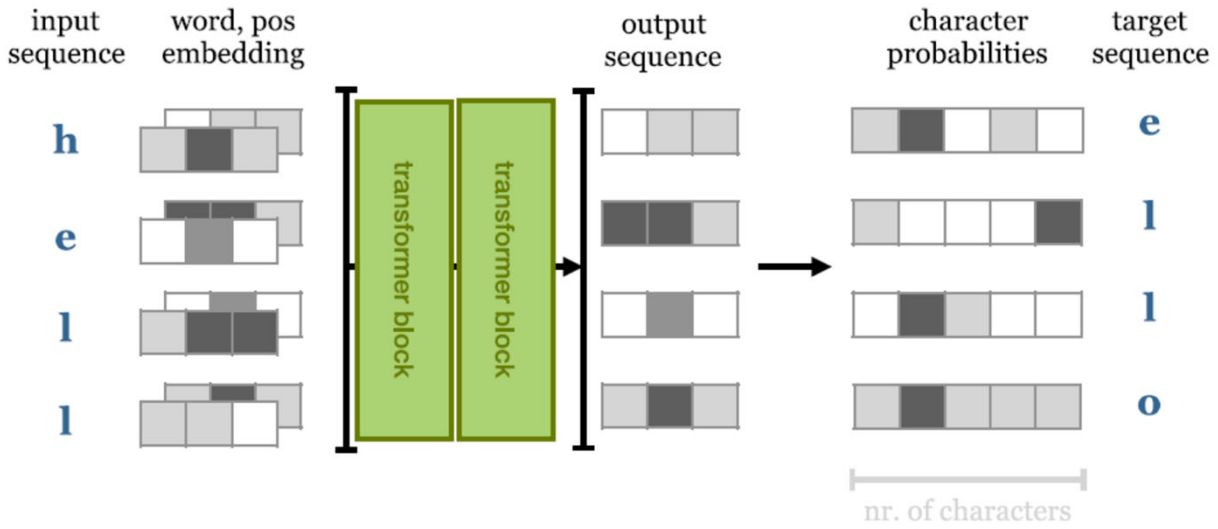
Transformer networks
Example: classifier (Bloem 2019)

- Binary classifier for word sequences
  - Targets: positive/ negative
  - The output sequence is averaged in order to produce a fixed size vector
  - Loss: cross entropy



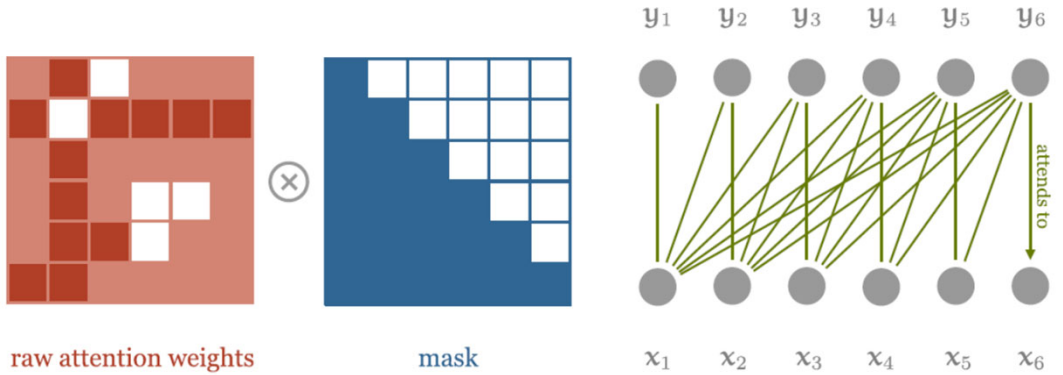
Transformer networks (Vaswani 2017, illustrations J. Alammar 2018, P. Bloem 2019)
Example: text generation transformer - autoregressive model

- Character level transformer for predicting next character from an input sequence
  - Input: a sequence
  - Output next character for each point in the sequence, i.e. language model
  - i.e. the target sequence is the input shifted one character to the left
  - Example with a words vocabulary



Transformer networks
Example: text generation transformer - autoregressive model (Bloem 2019)

- Because the transformer has access to the whole « h e l l » sequence, prediction for « e l l » becomes trivial
- If one wants to learn an autoregressive model one should prevent the transformer to look forward in the sequence
- Character level transformer for predicting next character from an input sequence
- For that one makes use of a **MASK** to the matrix of ot products before the softmax in the self attention module



Here x_i is the input in position i and y_i the output in position i

- Note: multiplication here is the elementwise multiplication

Transformer networks
Example: text generation transformer - autoregressive model (Bloem 2019)

- Example followed
- Training from sequences of length 256, using 12 transformer blocks and 256 embedding dimensions
- After training, let the model generate characters from a 256 input character sequence seed
  - For a sequence of 256 input characters the Transformer generates a distribution for the new character (257^{th}).
  - Sample from this distribution and feed back to the input for predicting the next (258^{th}) character, etc

Sample output (training from 10^8 characters from Wikipedia including markups):

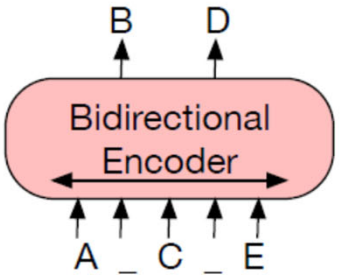
1228X Human & Rousseau. Because many of his stories were originally published in long-forgotten magazines and journals, there are a number of [[anthology|anthologies]] by different collators each containing a different selection. His original books have been considered an anthologie in the [[Middle Ages]], and were likely to be one of the most common in the [[Indian Ocean]] in the [[1st century]]. As a result of his death, the Bible was recognised as a counter-attack by the [[Gospel of Matthew]] (1177-1133),…

Large size transformers for NLP examples

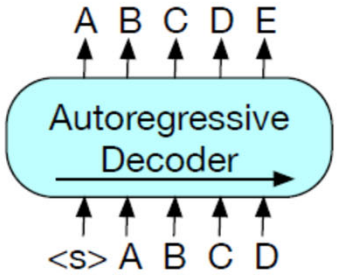
Families of transformer models: GPT

Large size language models based on transformers

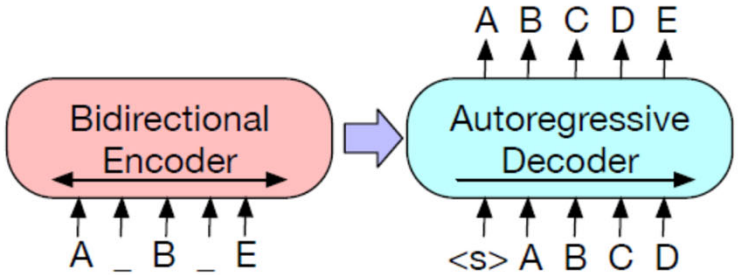
- Models architectures
  - Different schemes for using Transformers (figure from Lewis, et al. 2019. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension).



(a) BERT: Random tokens are replaced with masks, and the document is encoded bidirectionally. Missing tokens are predicted independently, so BERT cannot easily be used for generation.



(b) GPT: Tokens are predicted auto-regressively, meaning GPT can be used for generation. However words can only condition on leftward context, so it cannot learn bidirectional interactions.



(c) BART: Inputs to the encoder need not be aligned with decoder outputs, allowing arbitary noise transformations. Here, a document has been corrupted by replacing spans of text with a mask symbols. The corrupted document (left) is encoded with a bidirectional model, and then the likelihood of the original document (right) is calculated with an autoregressive decoder. For fine-tuning, an uncorrupted document is input to both the encoder and decoder, and we use representations from the final hidden state of the decoder.

Large size language models based on transformers
Decoders - GPT family (OpenAI)

- GPT (Radford et al. 2018), GPT 2 (Radford et al. 2019), GPT 3 (Radford et al. 2020), InstructGPT, ChatGPT, GPT4. etc
  - GPT means Generative Pre Training
  - Language models based on transformer **decoder** architecture (Liu et al. 2018)
    - As for the other Transformer models, training proceeds in 2 steps
      - **Unsupervised language modeling**
      - **Fine tuning on downstream tasks**
      - Successive models are larger and larger and trained on larger and larger corpora
  - GPT 2 comes in different versions from 117 M parameters (12 transformer decoder blocks) to 1.542 M parameters (48 transformer decoder blocks)
    - It is trained on a corpus of 8 M documents, 40 GB of text (scraped web pages curated by humans to ensure document quality)
    - Demonstrates the ability of language models to solve tasks they are not trained on
      - Hence proposes an alternative to fine tuning
  - GPT 3: 96 Transformer decoder modules stacked, 175 Billions parameters (2020)
    - 100 times bigger than GPT2
    - Demonstrates that very large models perform well on zero shot and few shot learning
    - Started developments by different companies on LLM, chatbots, programming tools, etc

Large size language models based on transformers
Decoders - GPT family (OpenAI) – GPT2

- GPT 2
  - Same general architecture than GPT with some modifications
    - layer normalization changed, initialization, scaling, etc…
  - Training dataset
    - 40 GB of text (scraped web pages curated by humans to ensure document quality)
  - Input representation
    - Modified Byte Pair Encoding (see later)
  - Training
    - Language model only (unsupervised)
  - Demonstrates that language models trained in an unsupervised way can achieve good performance, sometimes SOTA, on diverse tasks in few shot, zero shot learning schemes
  - Generalize the use of prompting for task conditioning and for providing few shots examples
    - Language allows to provide in a natural ways task indication and task examples
    - Translation: (translate to French, English text, French text)
    - Reading comprehension: (answer the question, document, question, answer)

Large size language models based on transformers
Decoders - GPT family (OpenAI) – GPT3

- GPT3 is 100 times larger than GPT2 – 175 B parameters for the larger model @year 2020
- Training dataset
  - Same as for GPT2 – about 3 B words cleaned and augmented
- Model
  - Same general architecture as GPT2 – auto-regressive decoder
- Demonstrates that very large models are able to perform SOTA on few shot and zero shot learning
  - Size change qualitatively the ability of the model: Emergence capacities with in-context learning
  - Starts the exploration of LLM for solving a variety of language tasks
  - At the core of later developments like ChatGPT

Large size language models based on transformers
GPT family (OpenAI) – GPT3

- Importance of size

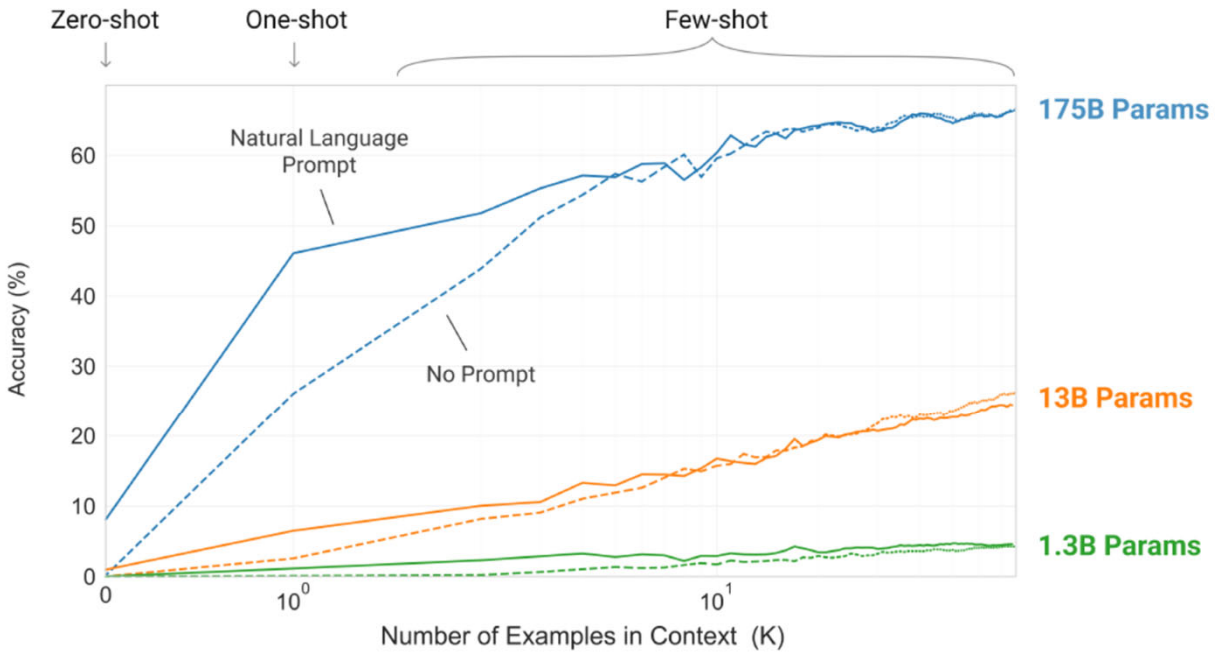


Figure 1.2: Larger models make increasingly efficient use of in-context information. We show in-context learning performance on a simple task requiring the model to remove random symbols from a word, both with and without a natural language task description (see Sec. 3.9.2). The steeper "in-context learning curves" for large models demonstrate improved ability to learn a task from contextual information. We see qualitatively similar behavior across a wide range of tasks.

Large size language models based on transformers
Decoders - GPT family (OpenAI) – GPT3

- Few shot, few shot learning

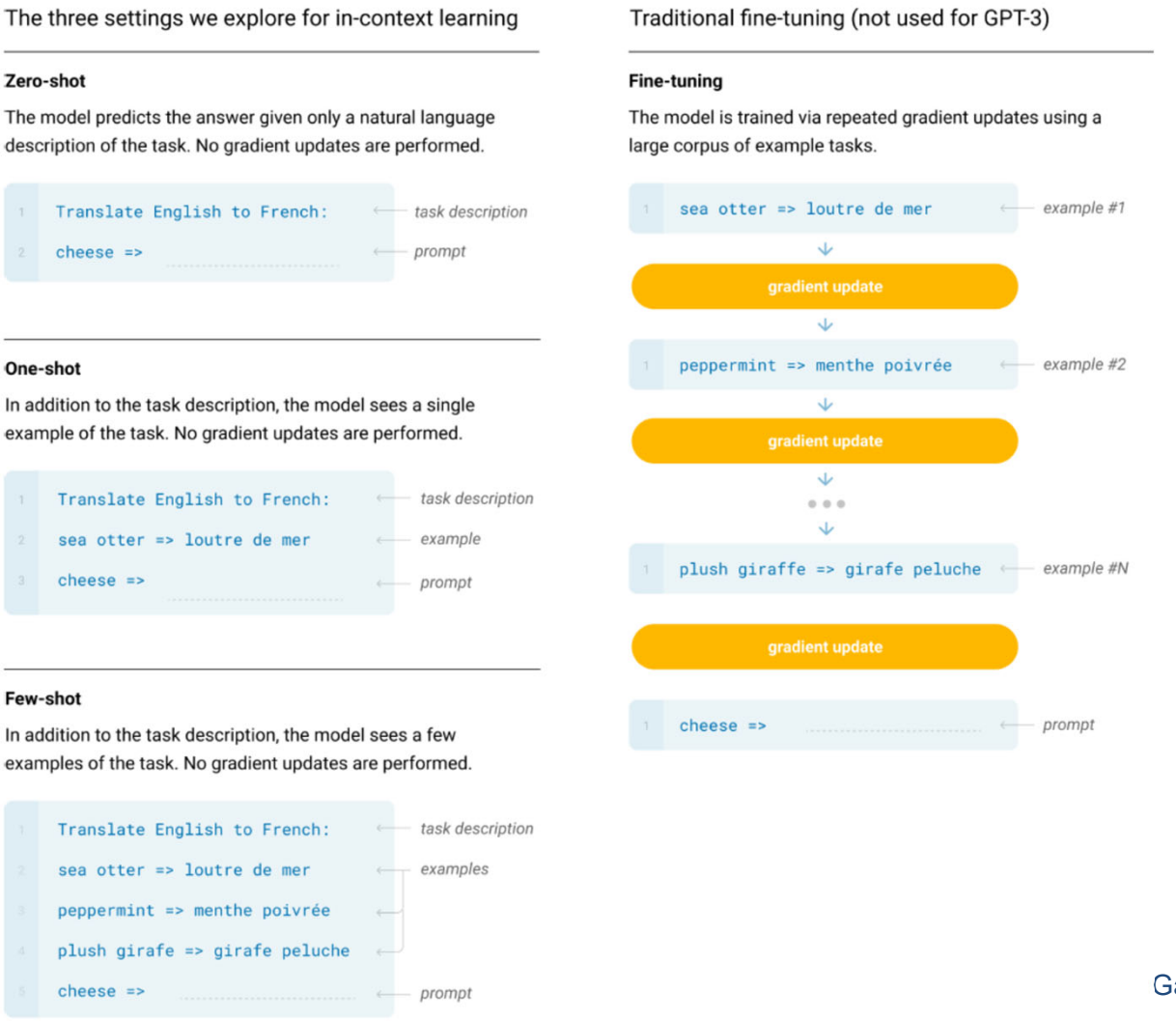


Figure 2.1: Zero-shot, one-shot and few-shot, contrasted with traditional fine-tuning. The panels above show four methods for performing a task with a language model – fine-tuning is the traditional method, whereas zero-, one-, and few-shot, which we study in this work, require the model to perform the task with only forward passes at test time. We typically present the model with a few dozen examples in the few shot setting.