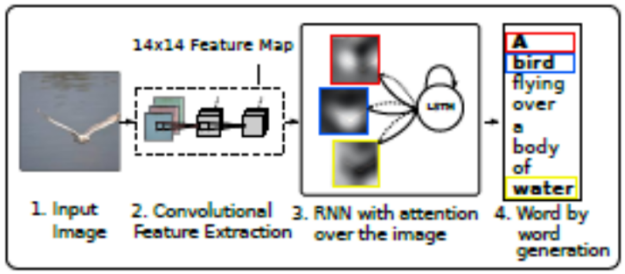
Attention Mechanism

Initial historical developments and examples

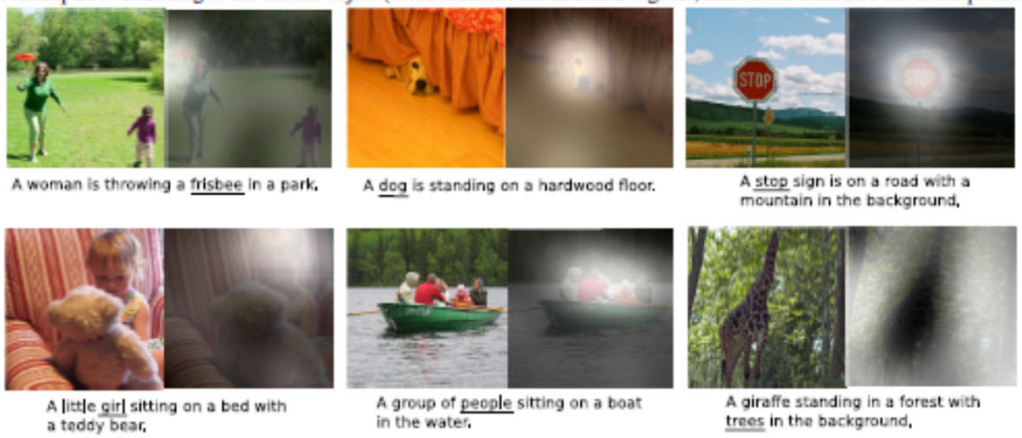
Attention mechanism

- Objective: focus on specific parts of the data representation for taking the current decision
  - Implemented as an additional differentiable modules in several architectures
- Illustration: attention on image while generating sentences

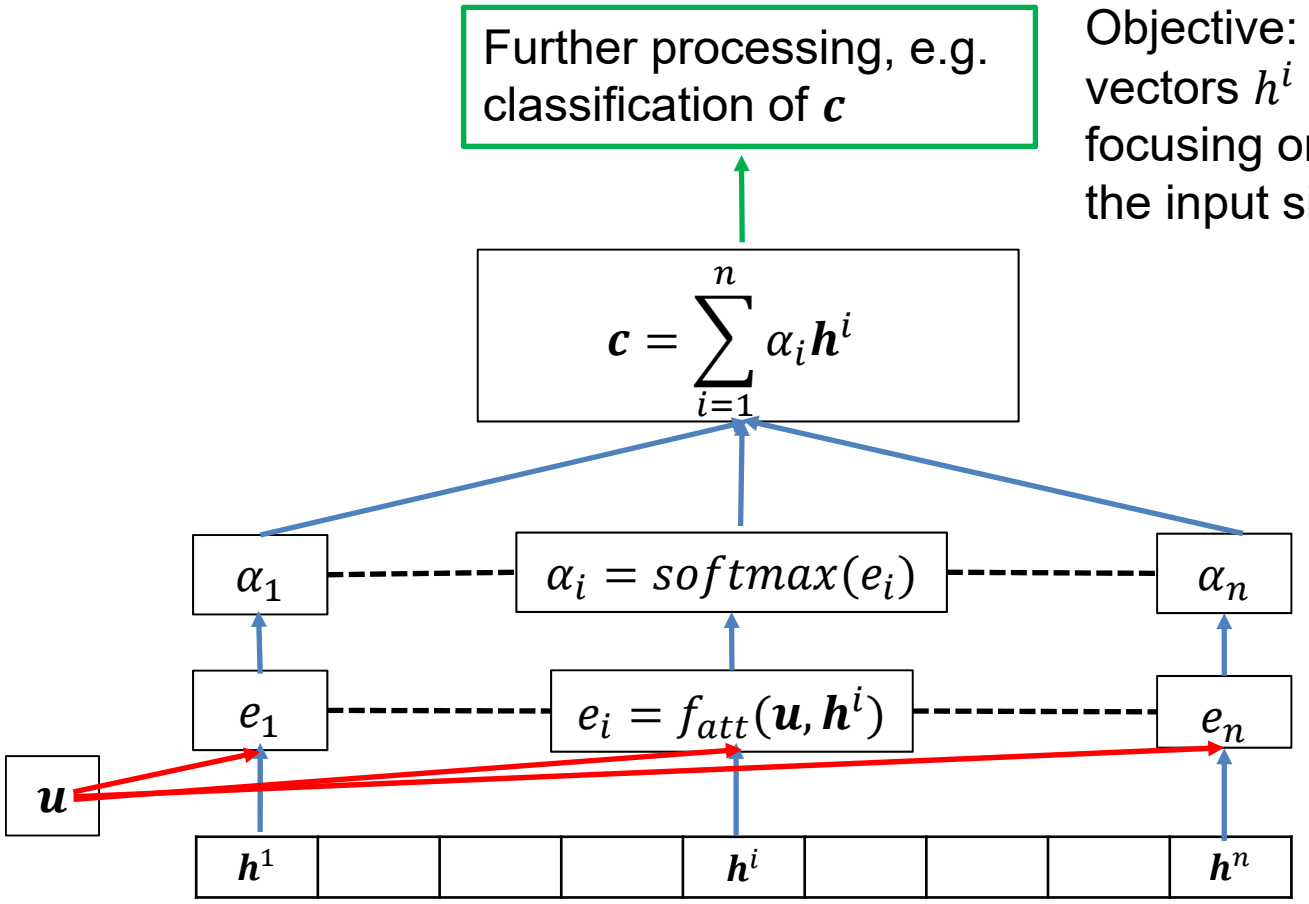


Figs. from Xu et al. 2015

Figure 4. Examples of attending to the correct object (*white* indicates the attended regions, *underlines* indicated the corresponding word)



Attention mechanism



Objective: learn a combination of input vectors h^i with attention weights focusing on the most relevant parts of the input signal $\boldsymbol{h}$

$\boldsymbol{h}^i \in R^d$: input, e.g. embedding or hidden output (e.g. RNN hidden layer)

$\boldsymbol{u} \in R^d$: additional info.

$\boldsymbol{c} \in R^d$: context vector

$e_i \in R$: attention factor

$\alpha_i \in R$: attention coefficient

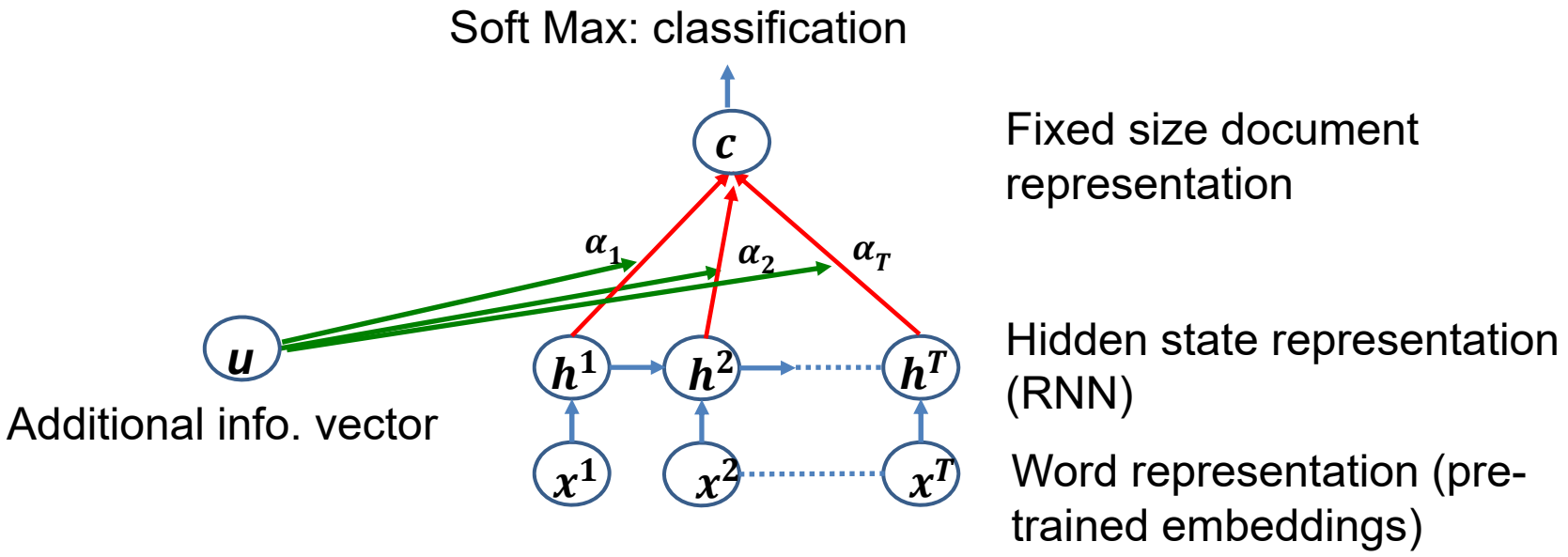
$$softmax(e_i) = \frac{\exp(e_i)}{\sum_{j=1}^{n} \exp(e_j)}$$

Attention mechanism

- Different attention functions f_{att}:
  - Additive
    - $f_{att}(\boldsymbol{u}, \boldsymbol{h}^i) = \boldsymbol{v}^T \tanh(W_1 \boldsymbol{h}^i + W_2 \boldsymbol{u}), \boldsymbol{v} \in R^d, \boldsymbol{h}^i \in R^d, W_1 : dxd, W_2 : dxd$
  - Multiplicative
    - $f_{att}(\boldsymbol{u}, \boldsymbol{h}^i) = \boldsymbol{u}^T W \boldsymbol{h}^i, \boldsymbol{u} \in R^d, W : dxd$
  - All the parameters (W, v, u) are learned
  - Many variants of these formulations

Attention mechanism
For document classification (adapted from Yang et al. 2016)

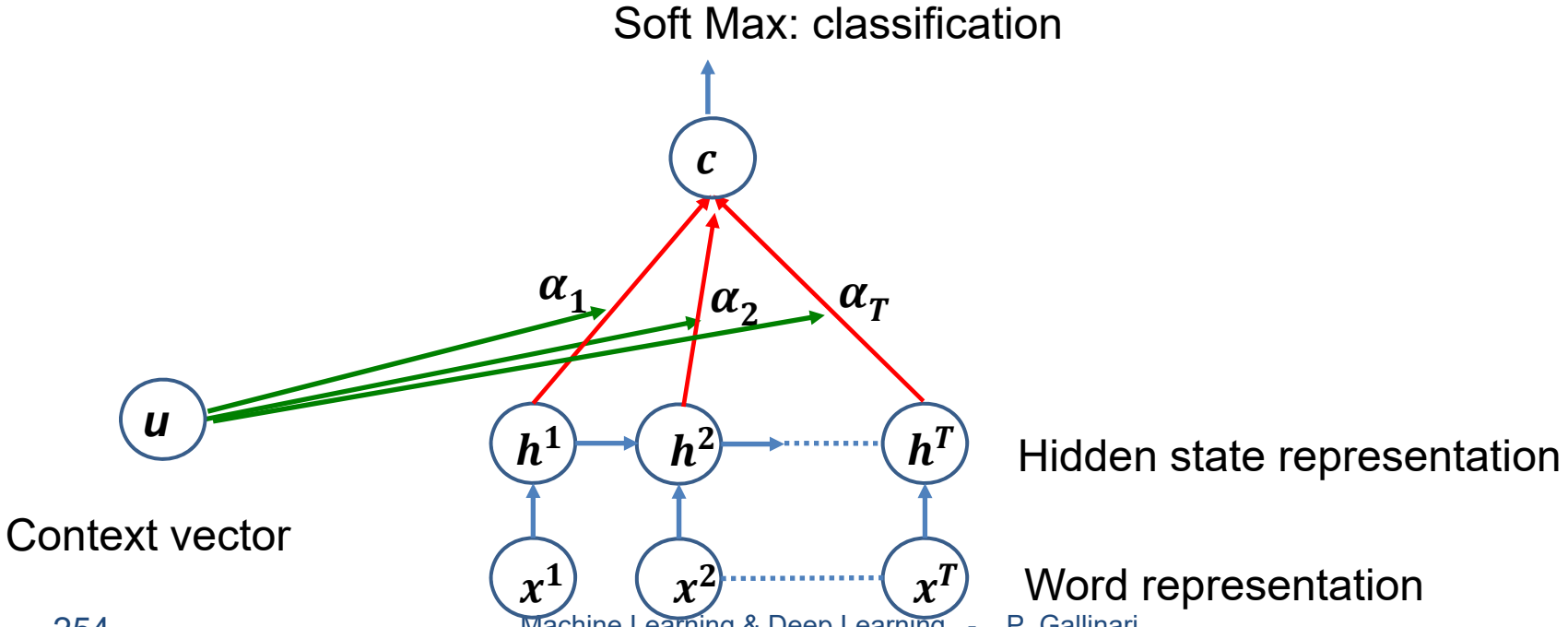
- Objective: classify documents using a sequential model of attention
  - Document : word sequence $w^1, \dots, w^T$
  - Objective: classify the document among predefined classes – learning criterion: log likelihood
  - Word sequence encodings (e.g. pretrained via Word2Vec): $x^1, \dots, x^T$
  - Corresponding hidden state sequence: $\boldsymbol{h}^1, \dots, \boldsymbol{h}^T$ obtained via a Recurrent NN



Attention mechanism
Example: document classification (adapted from Yang et al. 2016)

Parameters to be learned:
- Attention W, b, u
- Others: RNN, Softmax classifier

- $\boldsymbol{v}_j = \tanh\left(W\boldsymbol{h}^j + \boldsymbol{b}\right)$ (vector)
- $\alpha_j = \frac{\exp(\boldsymbol{v}_j.\boldsymbol{u})}{\sum_t \boldsymbol{v}_t.\boldsymbol{u}}$: attention weight (real value)
- $\boldsymbol{c} = \sum_{j=1}^{T} \alpha_j \boldsymbol{h}^j$: fixed size document representation (vector)
- $\boldsymbol{u}$: context vector to be learned (vector)



Attention mechanism
Example: document classification (adapted from Yang et al. 2016)

- Illustration (Yang et al. 2016)
  - Yelp reviews: ratings from 1 to 5 (5 is the best)
  - Classification = sentiment/ polarity classification
  - Hierarchical attention: word and sentence levels
  - Blue = word weight in the decision
  - Red = sentence weight in the decision (hierarchical attention model – 2 levels: sentences and words within a sentence)

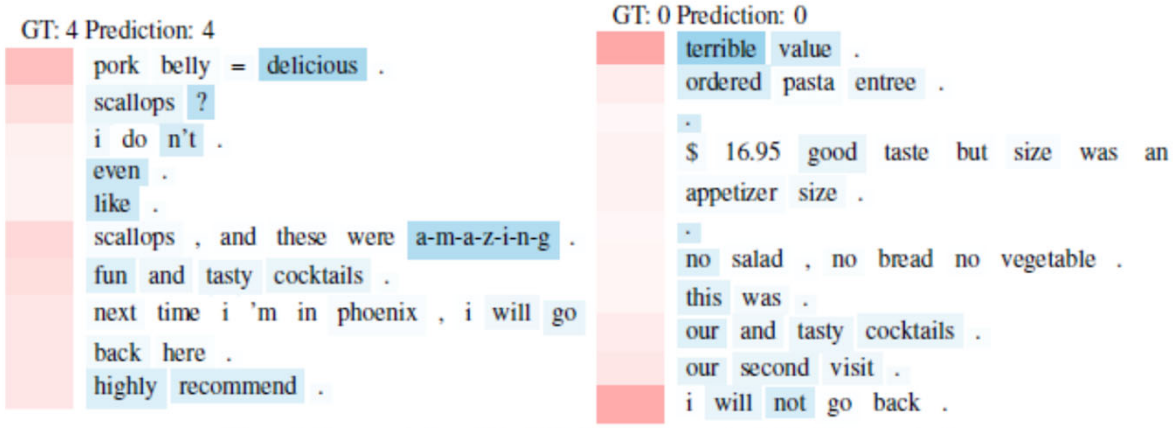


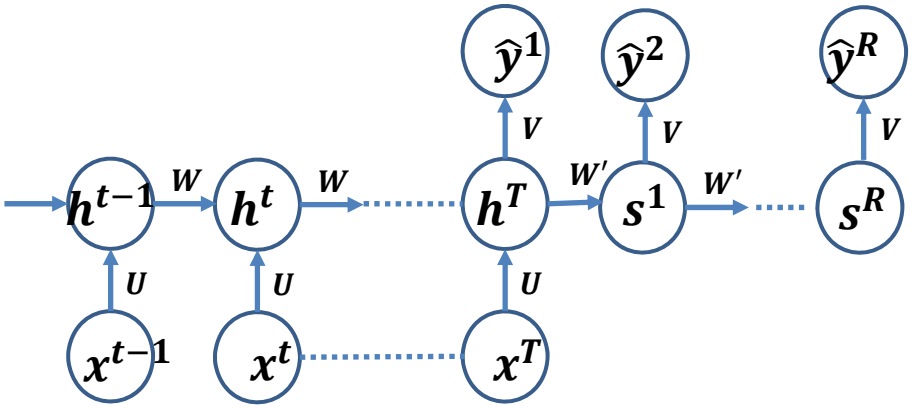
Figure 5: Documents from Yelp 2013. Label 4 means star 5, label 0 means star 1.

Attention mechanism for translation (adapted from Bahdanau et al. 2015 – initial introduction of attention)

- **Classical** Encoder – Decoder framework for translation
  - Encoder
    - Input sentence $\{\boldsymbol{x}^1, \dots, \boldsymbol{x}^T\}$ — word embeddings
    - Encoder: $\boldsymbol{h}^t = f_h(\boldsymbol{x}^t, \boldsymbol{h}^{t-1})$ — implemented via a RNN / LSTM
      - $\boldsymbol{h}^t$ is the hidden state for input $\boldsymbol{x}^t$
    - $\boldsymbol{c} = q(\boldsymbol{h}^1, \dots, \boldsymbol{h}^T)$ — for the original Encoder-Decoder framework, typically $\boldsymbol{c} = \boldsymbol{h}^T$ the last hidden state for the input sentence
  - Decoder
    - Output sentence $\{\boldsymbol{y}^1, \dots, \boldsymbol{y}^R\}$ — for simplification input and output sentence are taken at the same length
    - $p(\boldsymbol{y}^t | \boldsymbol{y}^1, \dots, \boldsymbol{y}^{t-1}, \boldsymbol{c}) = g(\boldsymbol{y}^{t-1}, \boldsymbol{s}^t, \boldsymbol{c})$ — implemented via a RNN or LSTM + softmax
      - $\boldsymbol{s}^t$ is the hidden state of the decoder for output $\boldsymbol{y}^t$
      - Decoding is conditionned on a unique vector $\boldsymbol{c}$ for the whole sentence

Attention mechanism
for translation (adapted from Bahdanau et al. 2015, initial introduction of attention)

- **Classical** Encoder – Decoder framework for translation

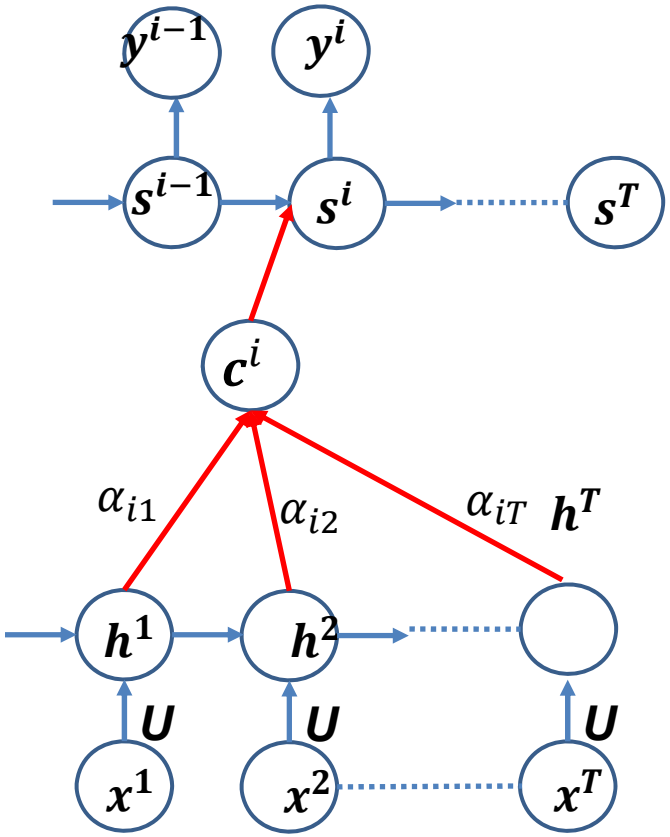


Attention mechanism for translation (adapted from Bahdanau et al. 2015, initial introduction of attention)

- **Attention mechanism**
  - Instead of conditionning the output $\boldsymbol{y}^i$ on the whole context $\boldsymbol{c} = \boldsymbol{h}^T$, the attention mechanism will use as context $\boldsymbol{c}_i$ a linear combination of the $\boldsymbol{h}^t, t = 1 \dots T$
    - One $\boldsymbol{c}_i$ is computed for each $\boldsymbol{y}^i$ instead of a common context $\boldsymbol{c}$ for all $\boldsymbol{y}^i$s
  - The encoder is the same as before
  - Decoder
    - $p(\boldsymbol{y}^i | \boldsymbol{y}^1, \dots, \boldsymbol{y}^{i-1}, \boldsymbol{x}) = g(\boldsymbol{y}^{i-1}, \boldsymbol{s}^i, \boldsymbol{c}_i)$
    - $\boldsymbol{s}^i = f(\boldsymbol{s}^{i-1}, \boldsymbol{y}^{i-1}, \boldsymbol{c_i})$
  - Context vector
    - $e_{ij} = a(\boldsymbol{s}^{i-1}, \boldsymbol{h}^j)$ computed via a simple MLP for example
    - $\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T} \exp(e_{ik})}$ weight of h^j when decoding y^i
    - $\boldsymbol{c}^i = \sum_{j=1}^{T} \alpha_{ij} \boldsymbol{h}^j$ context vector
- The whole system is trained end to end

Attention mechanism for translation (adapted from Bahdanau et al. 2015, initial introduction of attention)

- **Attention mechanism**



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

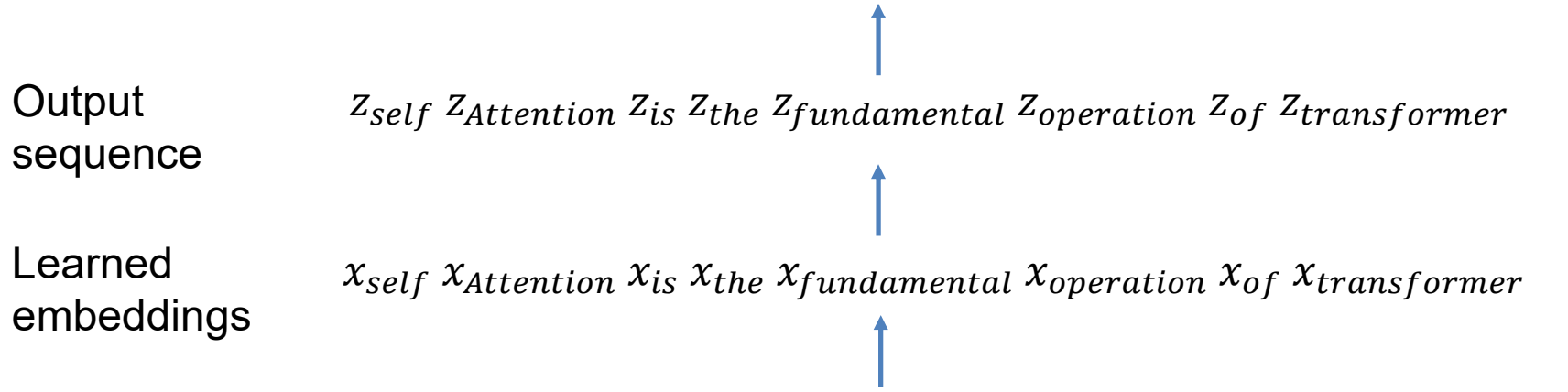
Transformer networks (Vaswani 2017, illustrations J. Alammar 2018-2019, P. Bloem 2019)
Self Attention

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



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

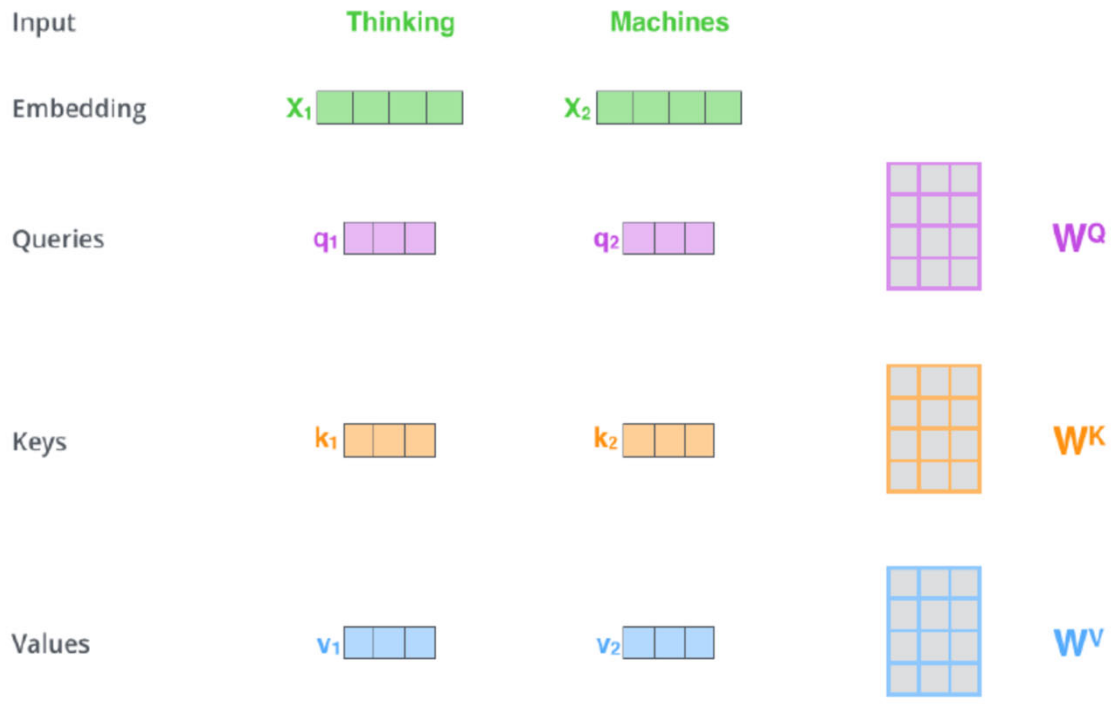
- Self attention is the only mechanism in the transformer that propagates information **between** vectors
- Any other operation is applied to each vector without interaction between vectors
- In the above example $z_{fundamental}$ is a weighted sum over all embedding vectors x weighted by their normalized 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



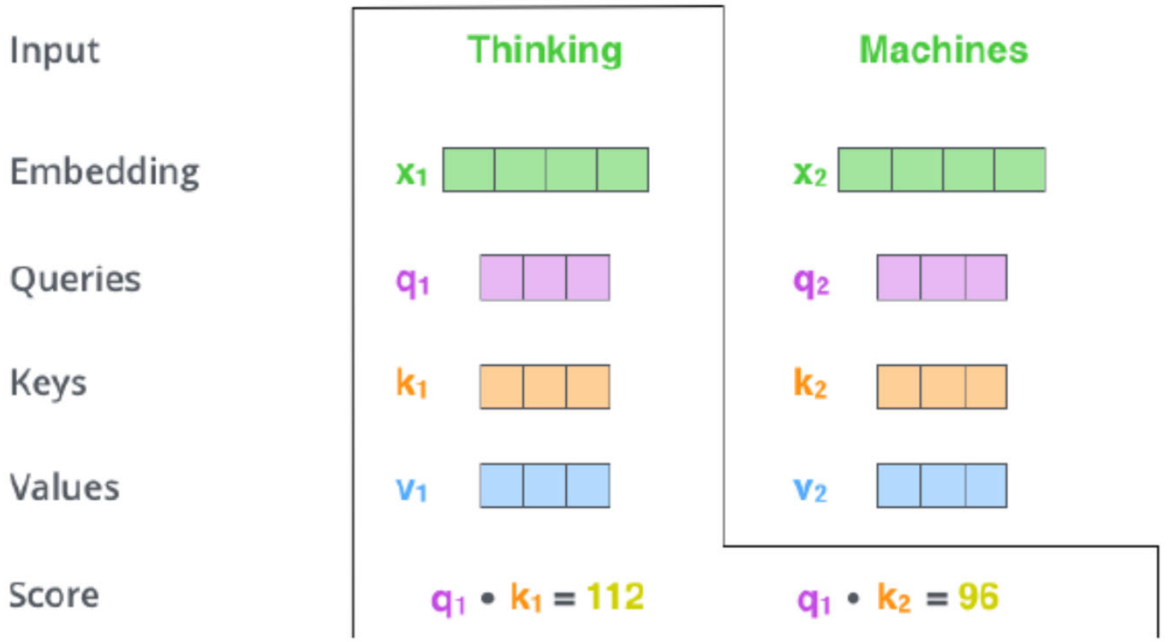
Multiplying x1 by the WQ weight matrix produces q1, the "query" vector associated with that word. We end up creating a "query", a "key", and a "value" projection of each word in the input sentence.

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**
    - $z_i = \sum_j \alpha_{ij} v_j$

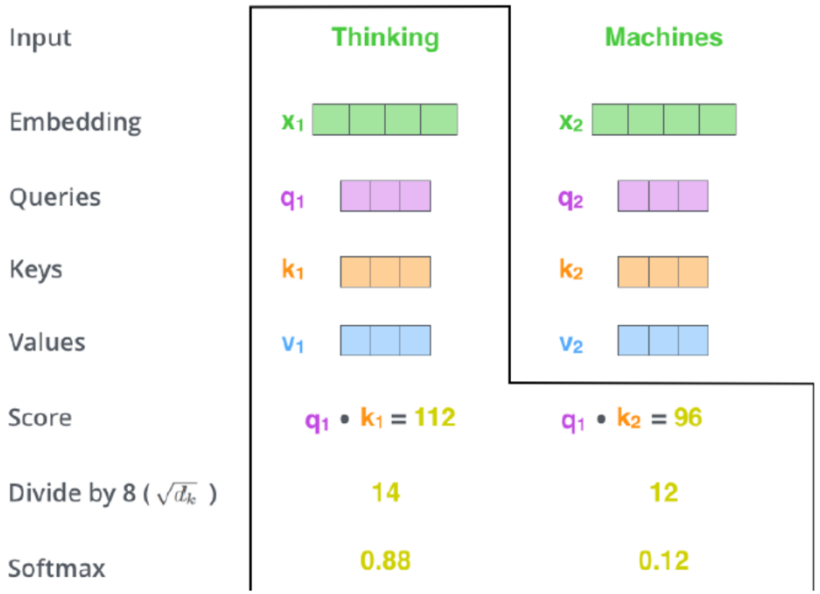
Transformer networks (Vaswani 2017, illustrations J. Alammar 2018, P. Bloem 2019) - – Queries, keys, values

- 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)



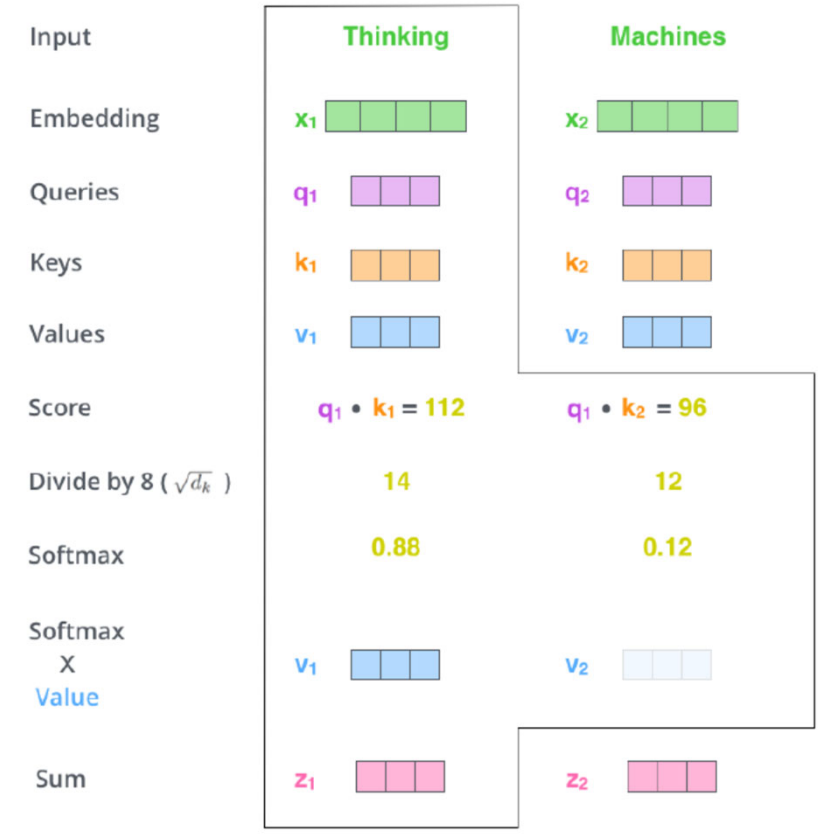
Transformer networks (Vaswani 2017, illustrations J. Alammar 2018, P. Bloem 2019) - – Queries, keys, values

- 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



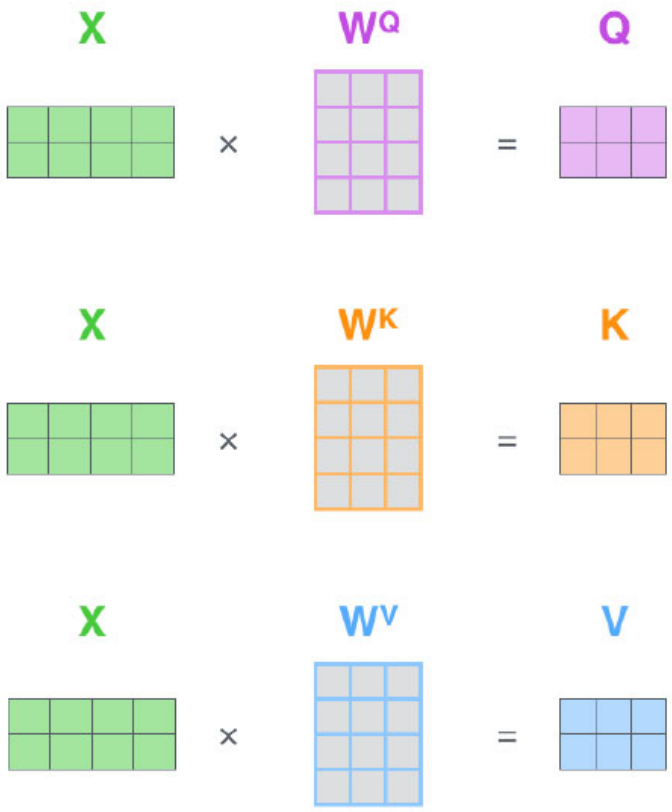
Transformer networks (Vaswani 2017, illustrations J. Alammar 2018, P. Bloem 2019) - – Queries, keys, values

- 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
  - $z_i = \sum_j \alpha_{ij} v_j$



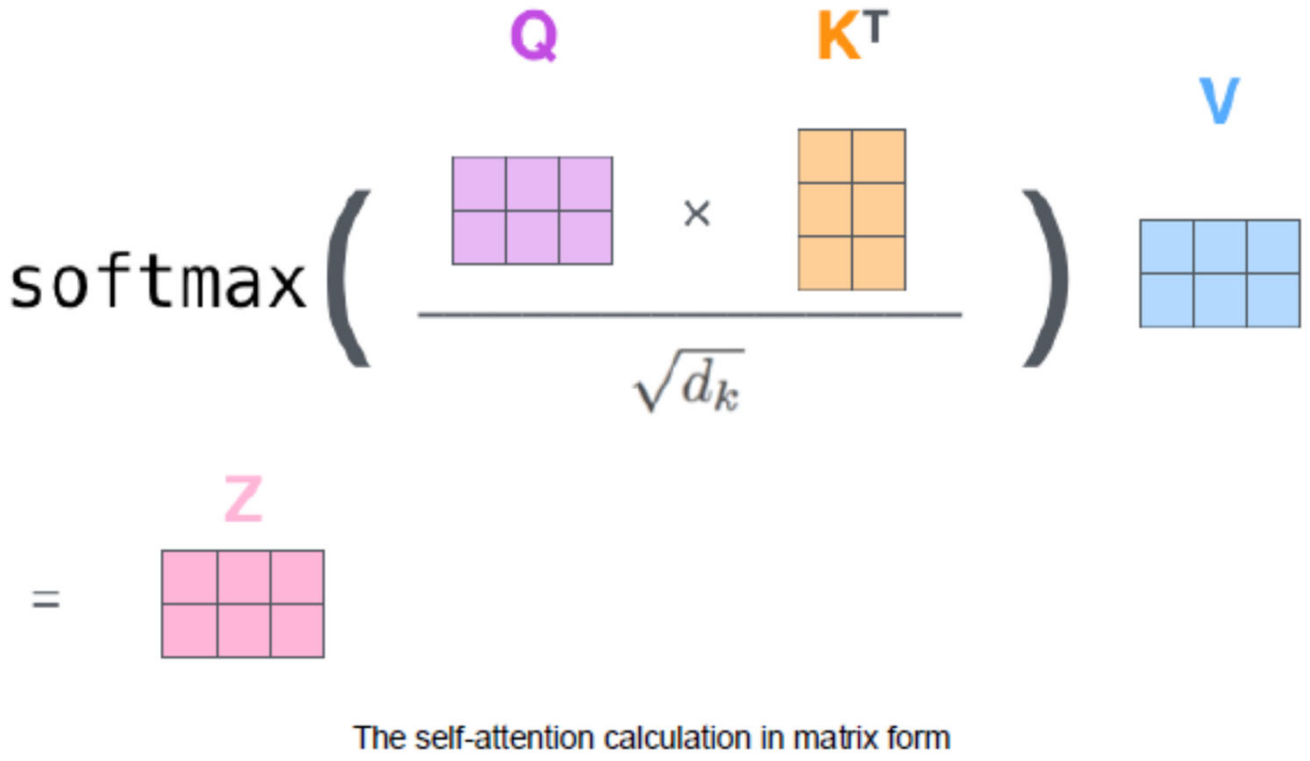
Transformer networks (Vaswani 2017, illustrations J. Alammar 2018, P. Bloem 2019) – Queries, keys, values

- In matrix form for our 2 words sentence



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

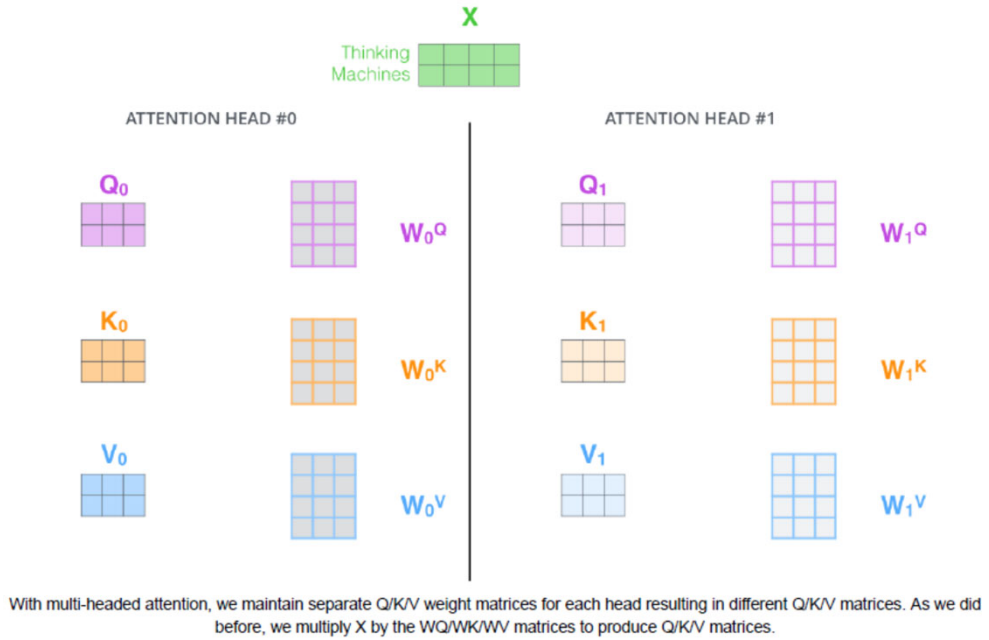
- Compute the output of the self attention layer at position 1
  - Matrix form



The self-attention calculation in matrix form

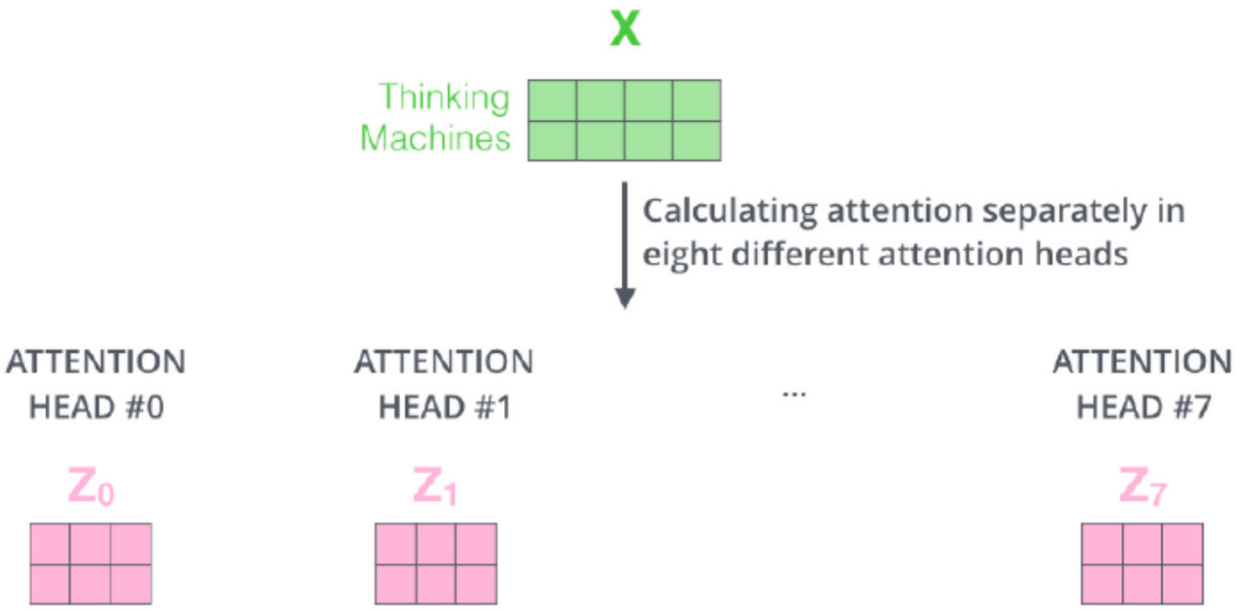
Transformer networks (Vaswani 2017, illustrations J. Alammar 2018, P. Bloem 2019)) - – Queries, keys, values

- 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

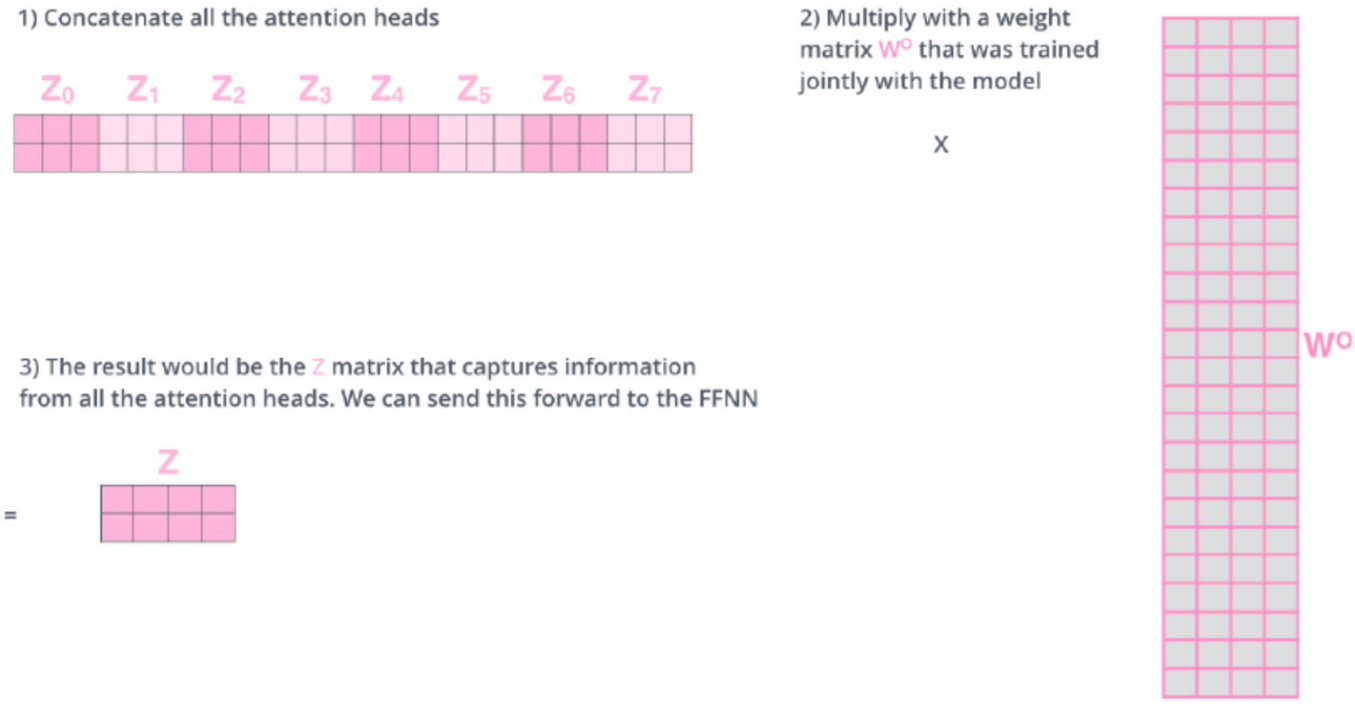


Transformer networks (Vaswani 2017, illustrations J. Alammar 2018, P. Bloem 2019))

- 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

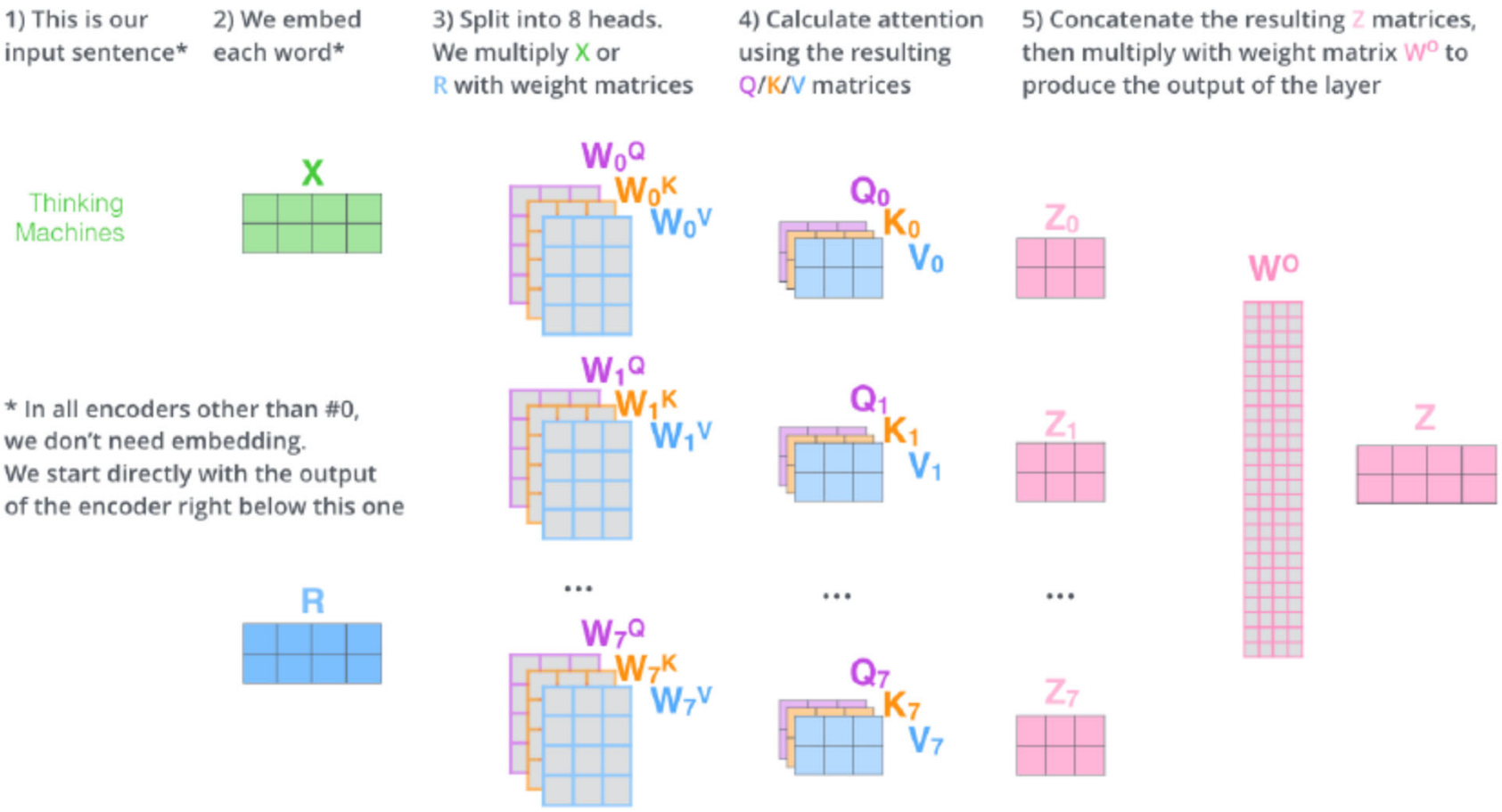
Transformer networks (Vaswani 2017, illustrations J. Alammar 2018, P. Bloem 2019)

- 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



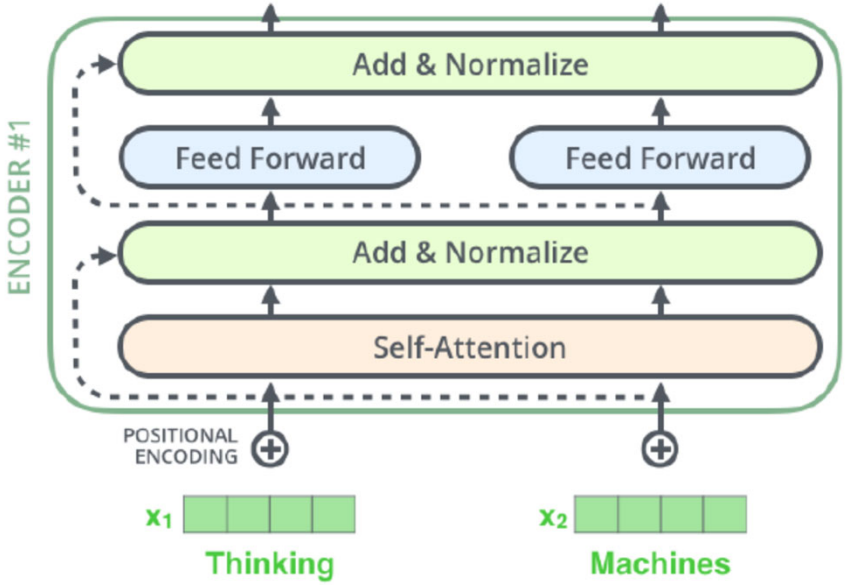
Transformer networks (Vaswani 2017, illustrations J. Alammar 2018, P. Bloem 2019)

- 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

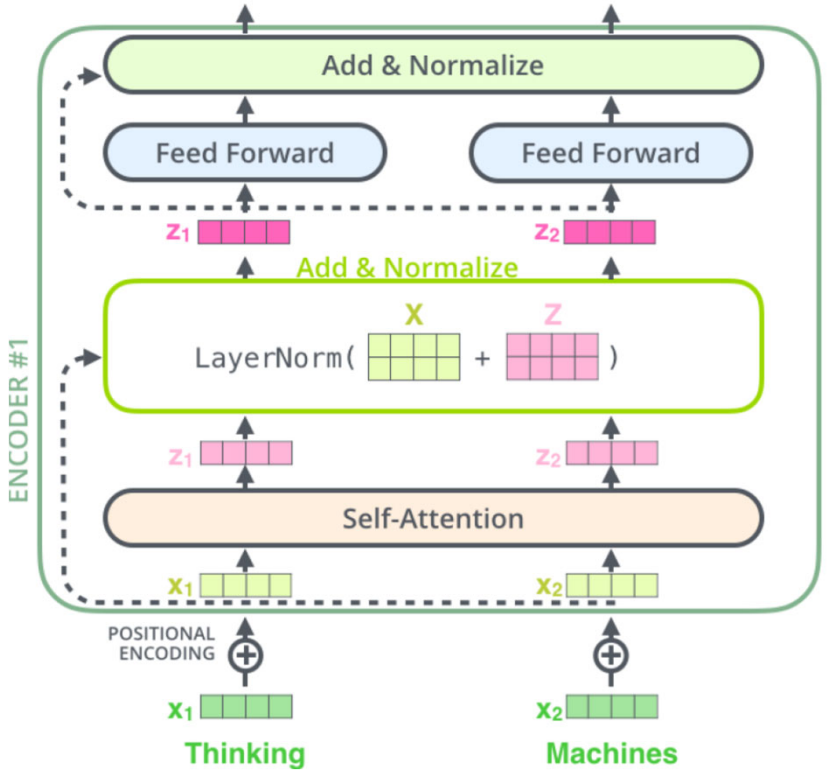


- Normalization layers (layer normalization)
- Multiple self attention modules per encoder
- Residual (skip) connections like in ResNet (see dashes --->)
- Positional encoding

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

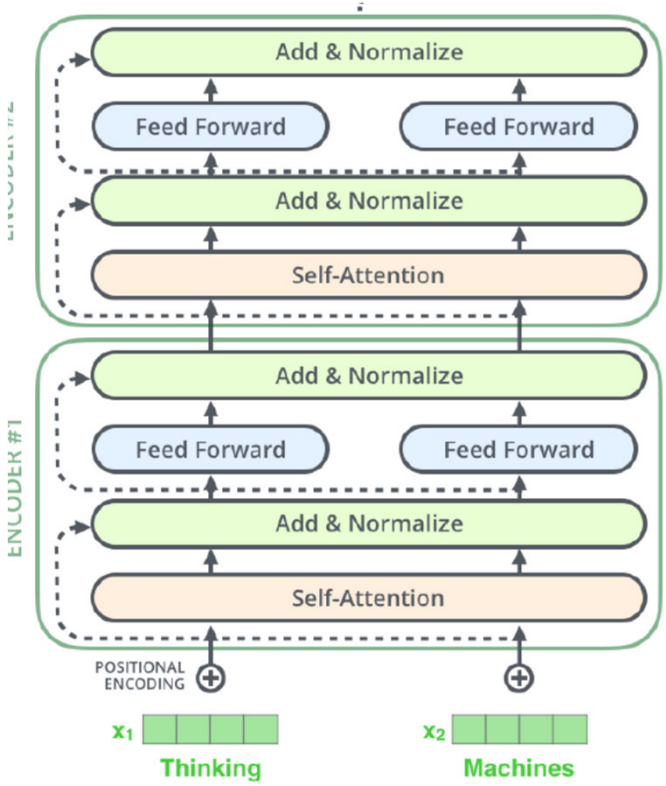
Transformer networks (Vaswani 2017, illustrations J. Alammar 2018, P. Bloem 2019)
Transformer module

- 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

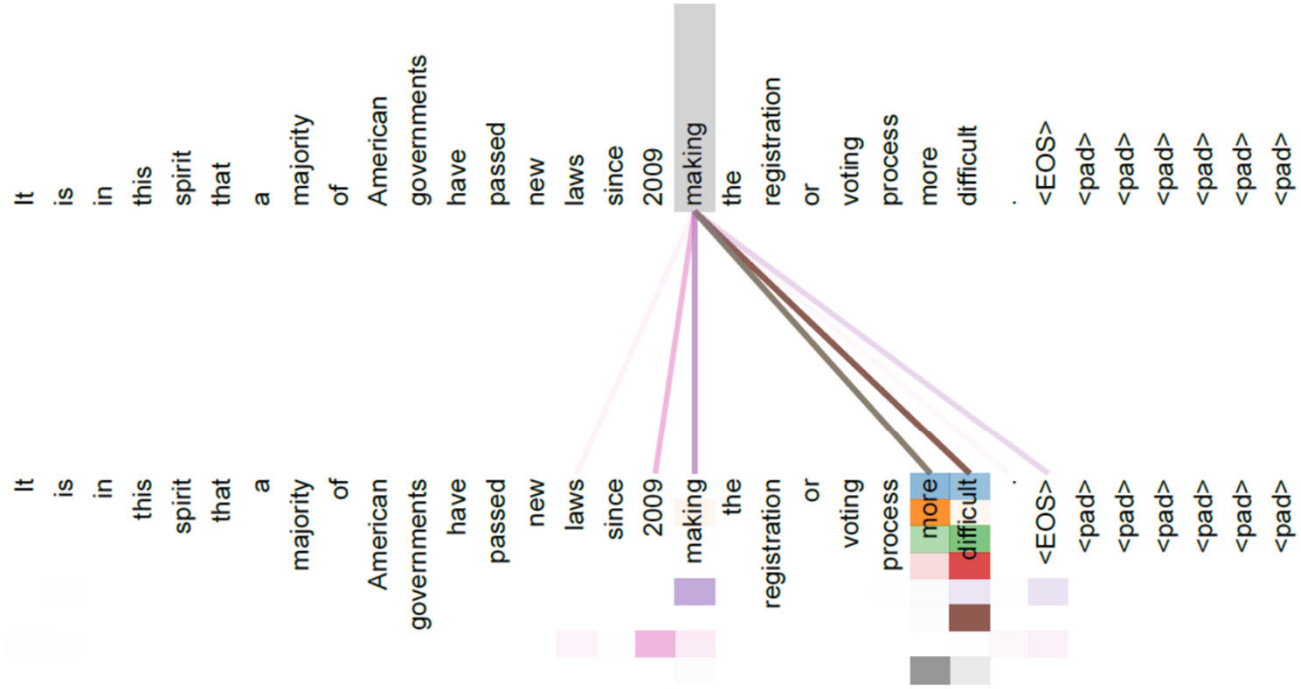


Figure 3: An example of the attention mechanism following long-distance dependencies in the encoder self-attention in layer 5 of 6. Many of the attention heads attend to a distant dependency of the verb 'making', completing the phrase 'making...more difficult'. Attentions here shown only for the word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

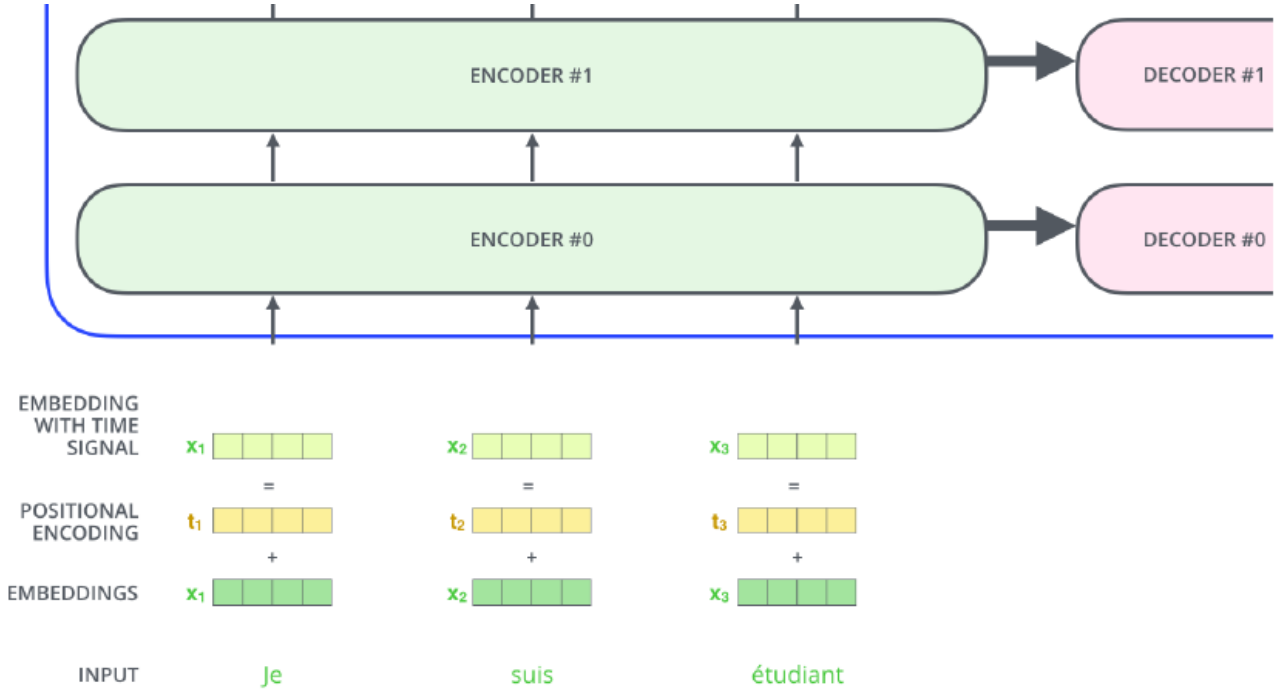
Fig. (Vaswani 2017)

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

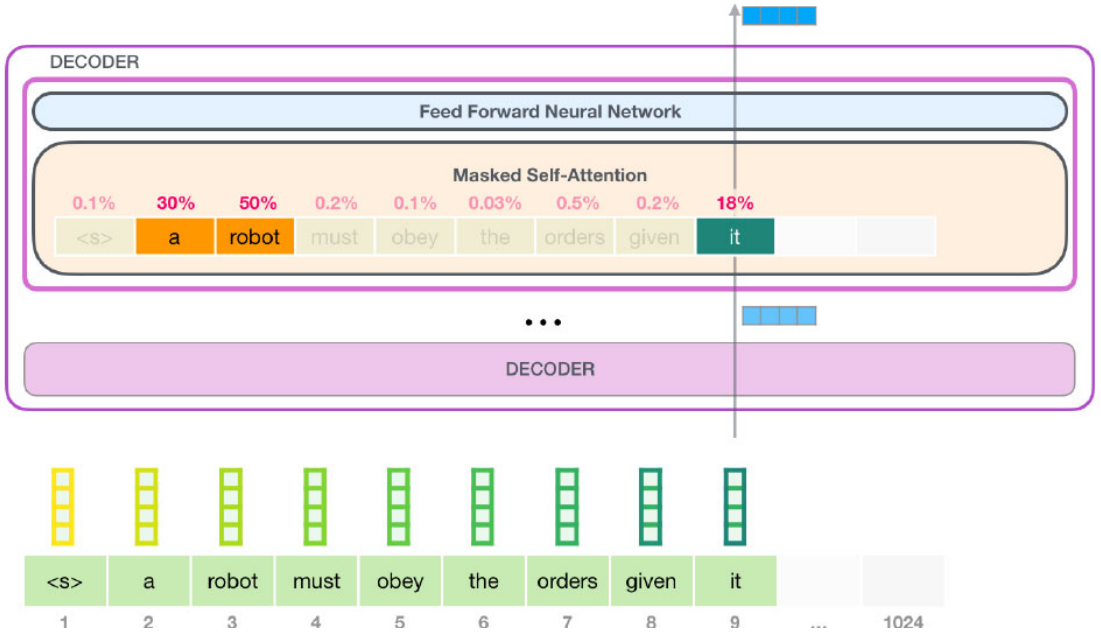
Transformer networks (Vaswani 2017, illustrations J. Alammar 2018-2019, P. Bloem 2019)

- Positional encoding



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

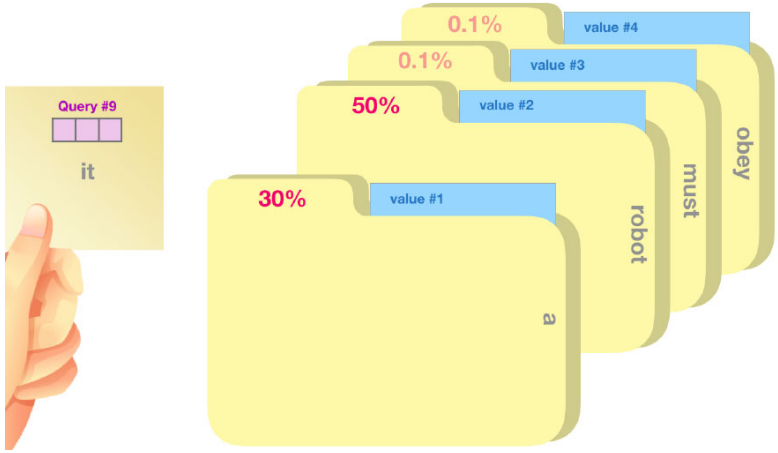
- Intuition on the Query/Key/value components (J. Alammar 2019)
- Consider the sentence
  - « a robot must obey the orders given it by human beings … »
  - « It » refers to « a robot »
    - This is what self attention should detect
  - Consider self attention in the decoder module when processing the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



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

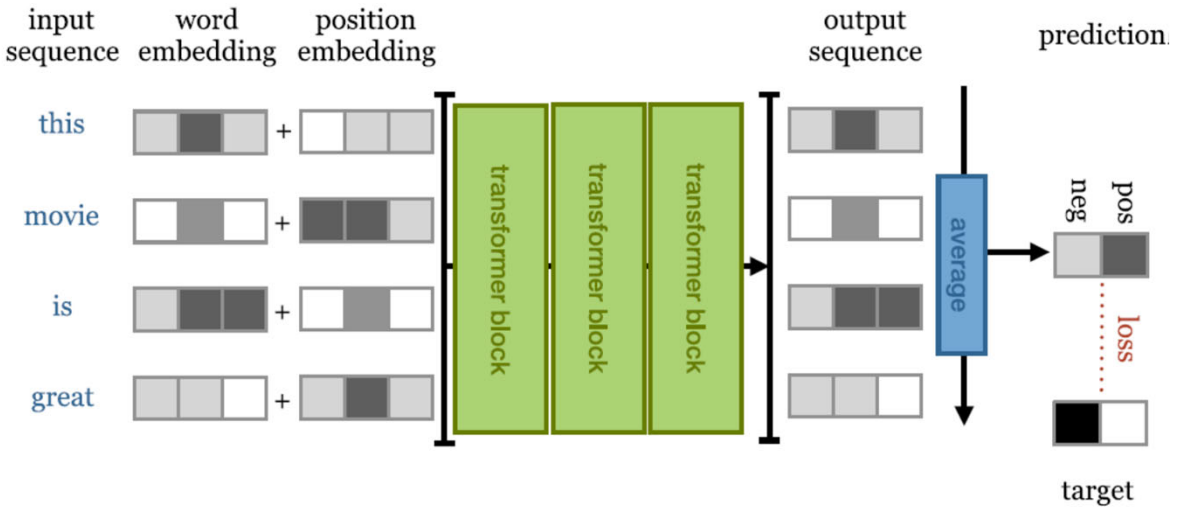
- Intuition on the Query/Key/value components (J. Alammar 2019)
  - The query is a representation of the current word used to score against all the other words (using their keys). We only care about the query of the token we're currently processing.
  - Key vectors are like labels for all the words in 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.



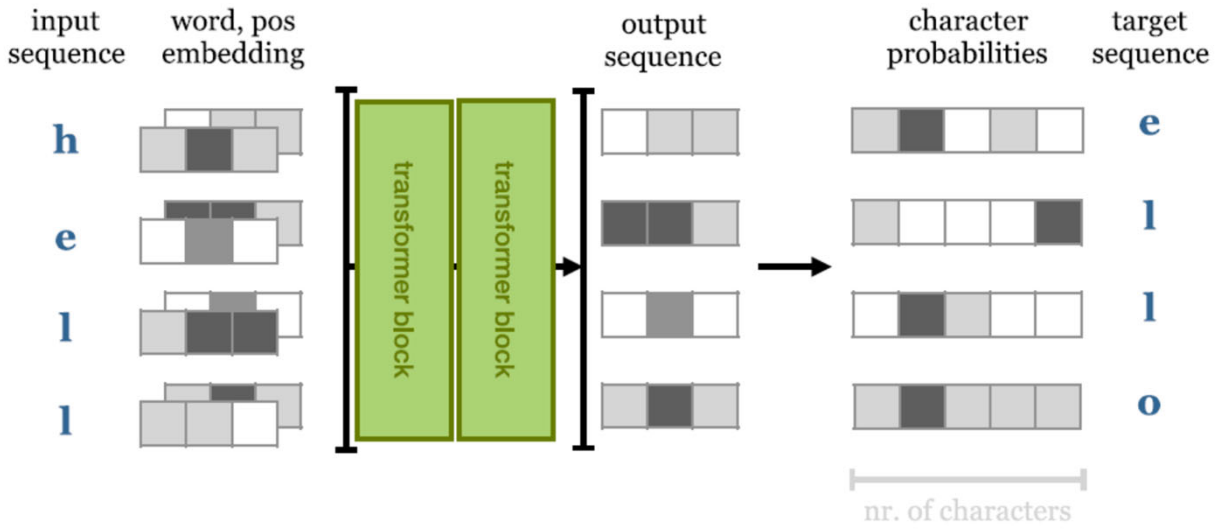
Transformer networks
Example: classifier (Bloem 2019)

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



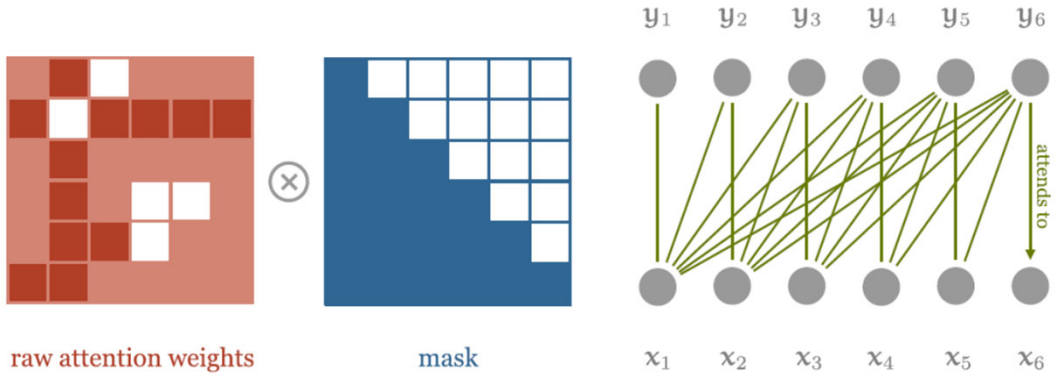
Transformer networks (Vaswani 2017, illustrations J. Alammar 2018, P. Bloem 2019)
Example: text generation transformer - autoregressive model

- 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)

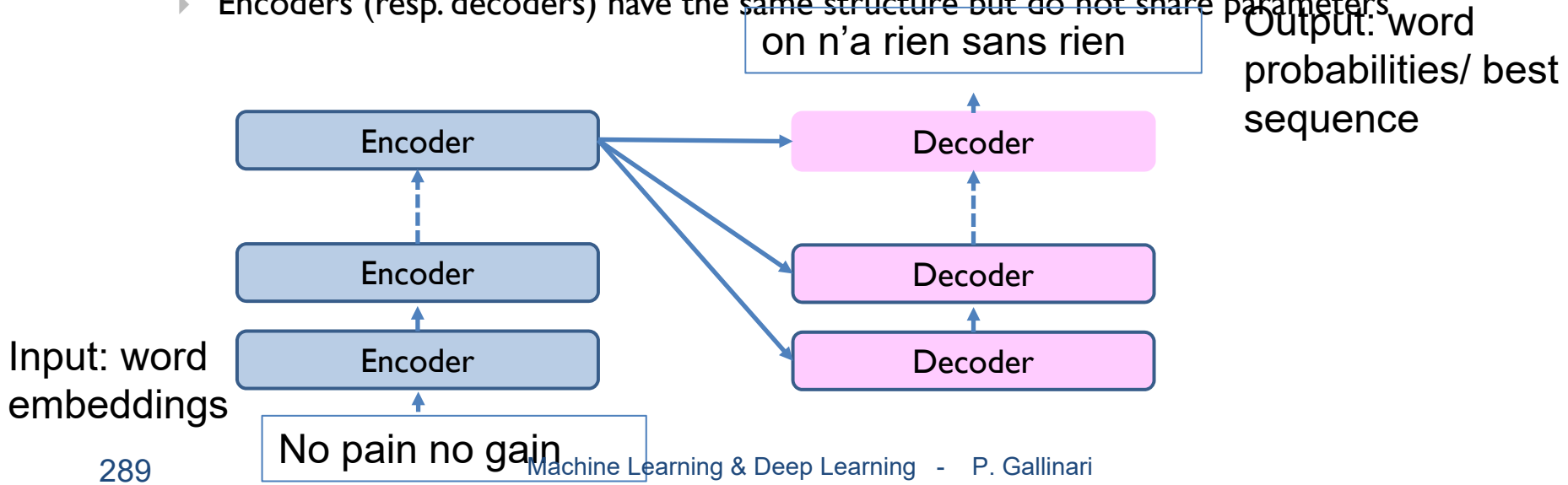
- Example followed
- Training from sequences of length 256, using 12 transformer blocks and 256 embedding dimensions
- After training, let the model generate characters from a 256 input character sequence seed
  - For a sequence of 256 input characters the Transformer generates a distribution for the 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),…

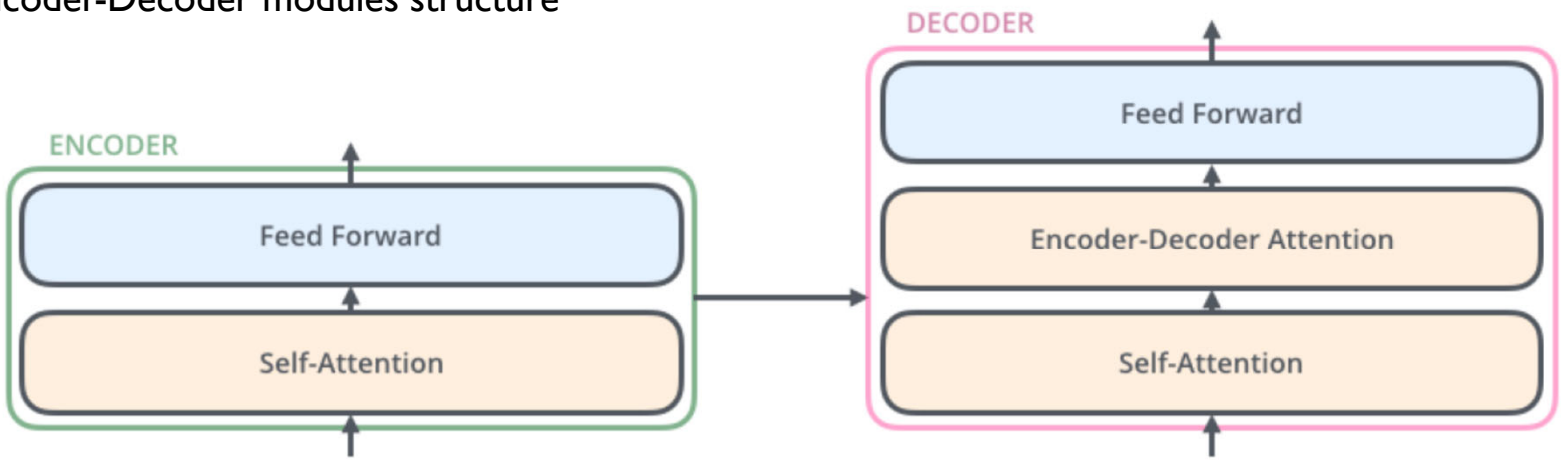
Appendix - Historical side: Transformer networks (Vaswani 2017)

- The first implementation of Transformer was proposed by (Vaswani 2017) as an encoder-decoder scheme
- Modern implementation make use of transformer blocks, either encoders, decoders or encoder-decoder schemes
- It is however interesting to look at the initial idea in order to understand the vocabulary
- General scheme
  - Stacks of encoder/ decoder modules
  - Encoders (resp. decoders) have the same structure but do not share parameters



Appendix - Historical side: Transformer networks (Vaswani 2017, illustrations J. Alammar 2018)

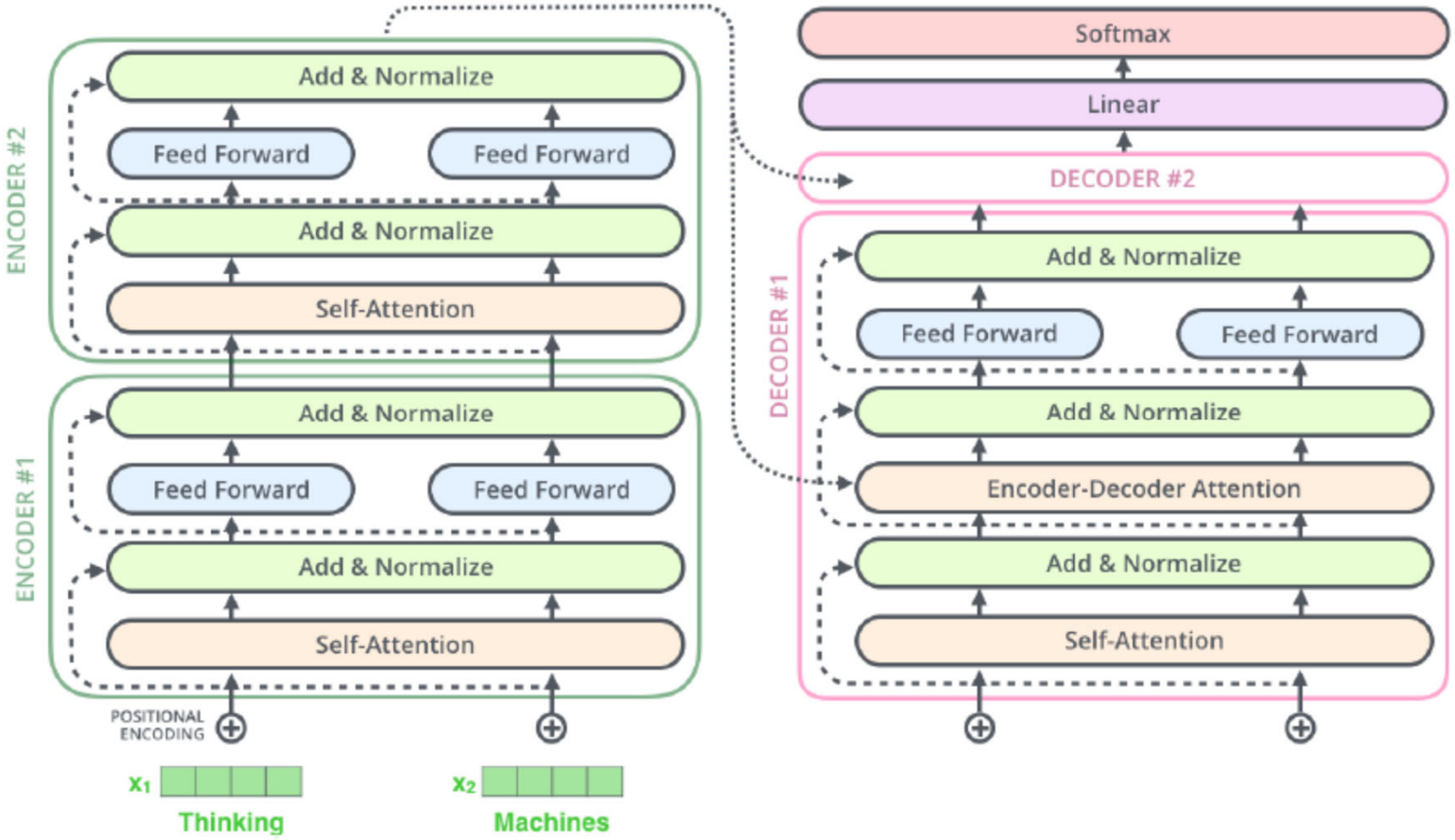
- Encoder-Decoder modules structure



- Encoder
  - Input flows through a self attention layer – encoding of a word in the sequence will depend on the other words
  - Outputs of the self attention layer are fed in a feed-forward NN. The same network is used for each word position
- Decoder: 2 differences with the encoder
  - 1. The decoder has an additional encoder-decoder attention layer that focuses on relevant parts of the input provided by the encoder (when the self attention module below it looks at the info from the lower layer of the decoder).
  - 2. For the self attention module, the decoder can only look at past information to predict the next word – this is similar to the autoregressive example seen before

Appendix - Historical side: Transformer networks (Vaswani 2017) illustration: J. Alammar 2018

- Encoder + Decoder modules



Appendic - Historical side: Transformer networks (Vaswani 2017) illustration: J. Alammar 2018

- Modern architectures use either encoder (BERT), decoder (GPT) or encoder-decoder (T5) schemes
  - BERT (Google) makes use of masked inputs (more on that later) and looks at the full input sequence
  - GPT (Open AI) is an autoregressive model (like a classical language model) and looks only at past items for predicting the future
  - T5 (Google) is an encoder-decoder model designed for reformulating several NLP tasks in a text to text framework

Large size transformers examples

Contextual encodings: ELMo

Large size SOTA Transformer models: GPT – Decoder model

BERT – encoder model

T5 – Encoder Decoder model

Large size transformers
Some resources

- HuggingFace Transformer library
  - Offers several implementation of recent transformer models in PyTorch and Tensorflow
    - https://huggingface.co/
  - List of transformers from Huggingface
    - https://huggingface.co/docs/transformers/index
    - https://huggingface.co/models
  - BERT
  - Tutorial on BERT word embeddings https://mccormickml.com/2019/05/14/BERT-word-embeddings-tutorial/
  - BERT as used in Google search engine as of 2019
    - https://searchengineland.com/faq-all-about-the-bert-algorithm-in-google-search-324193#:~:text=BERT%2C%20which%20stands%20for%20Bidirectional,of%20words%20in%20search%20queries.
- Demos for different NLP tasks from Allen AI
  - https://demo.allennlp.org/
  - For a GPT2 demo see « language modeling »

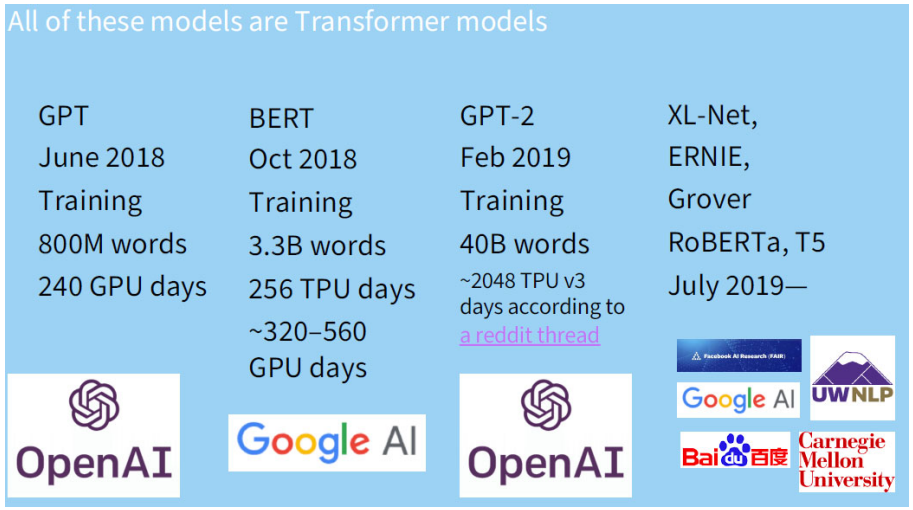
Large size transformers
Teaser

- NLP
  - ChatGPT (OpenAI) https://chat.openai.com/chat
  - LaMDA - https://blog.google/technology/ai/lamda/, https://arxiv.org/abs/2201.08239
  - PALM - https://ai.googleblog.com/2022/04/pathways-language-model-palm-scaling-to.html

- Text to Image
  - Craiyon : public version of Dall-E - https://www.craiyon.com/
  - Dall-e https://openai.com/blog/dall-e/, https://openai.com/dall-e-2/

Large size language models based on transformers

- Right after the seminal publication on transformers (Vaswany 2027), several large size models based on these ideas were developed by different groups



- They have in common:
  - Large size models and large corpora!!
  - Credo:
    - pretrain on large size corpora and fine tune on downstream tasks - Larger is better ☹
  - Training on very large size corpora
    - General objective: learn token representations in an unsupervised way from large corpora that could be used with little adaptation for specific downstream tasks (requiring « small » labeled datasets) w/ or w/o fine tuning of the whole model
  - Easily adaptable for a variety of downstream tasks
    - Token level e.g. Named Entity Recognition (NER), …
    - Sentence level e.g. Query Answering Q/A, text classification, …

Large size language models based on transformers

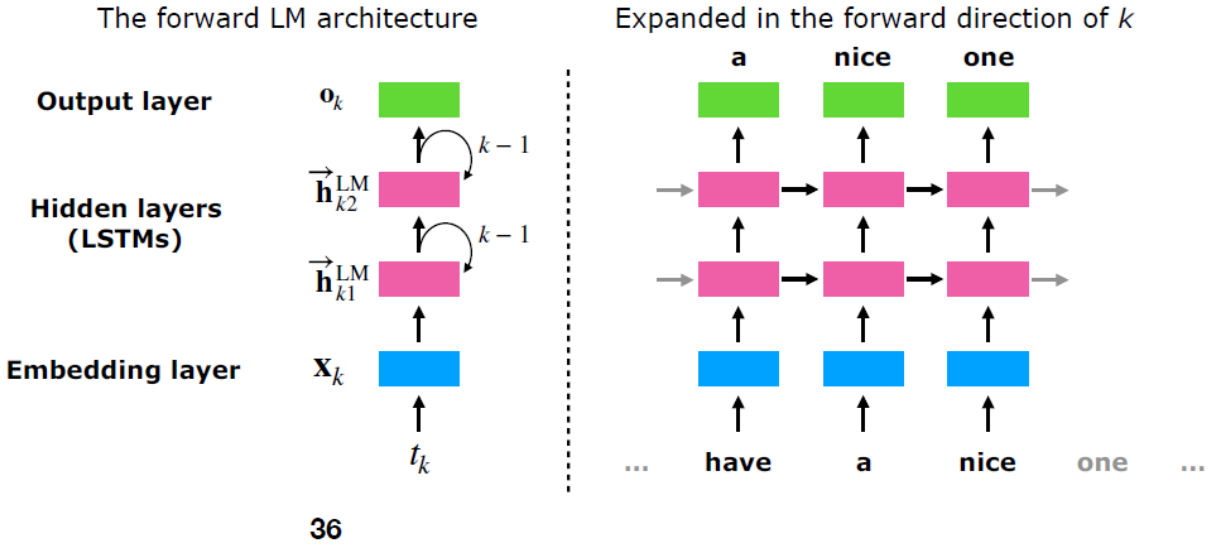
ELMo (Peters et al. 2018. Deep contextualized word representations. *NAACL* (2018)).

- Contextual word representation
  - In Word2Vec, FastText, GloVe, word representations are unique
  - We might want context dependent word representations
  - This is what ELMo introduced
  - (slides from https://fr2.slideshare.net/shuntaroy/a-review-of-deep-contextualized-word-representations-peters-2018)

- Embeddings from Language Models: **ELMo**
- Learn word embeddings through building *bidirectional language models* (biLMs)
  - biLMs consist of forward and backward LMs
    - Forward: $p(t_1, t_2, \ldots, t_N) = \prod_{k=1}^{N} p(t_k \mid t_1, t_2, \ldots, t_{k-1})$
    - Backward: $p(t_1, t_2, \ldots, t_N) = \prod_{k=1}^{N} p(t_k \mid t_{k+1}, t_{k+2}, \ldots, t_N)$

Large size language models based on transformers
ELMo -

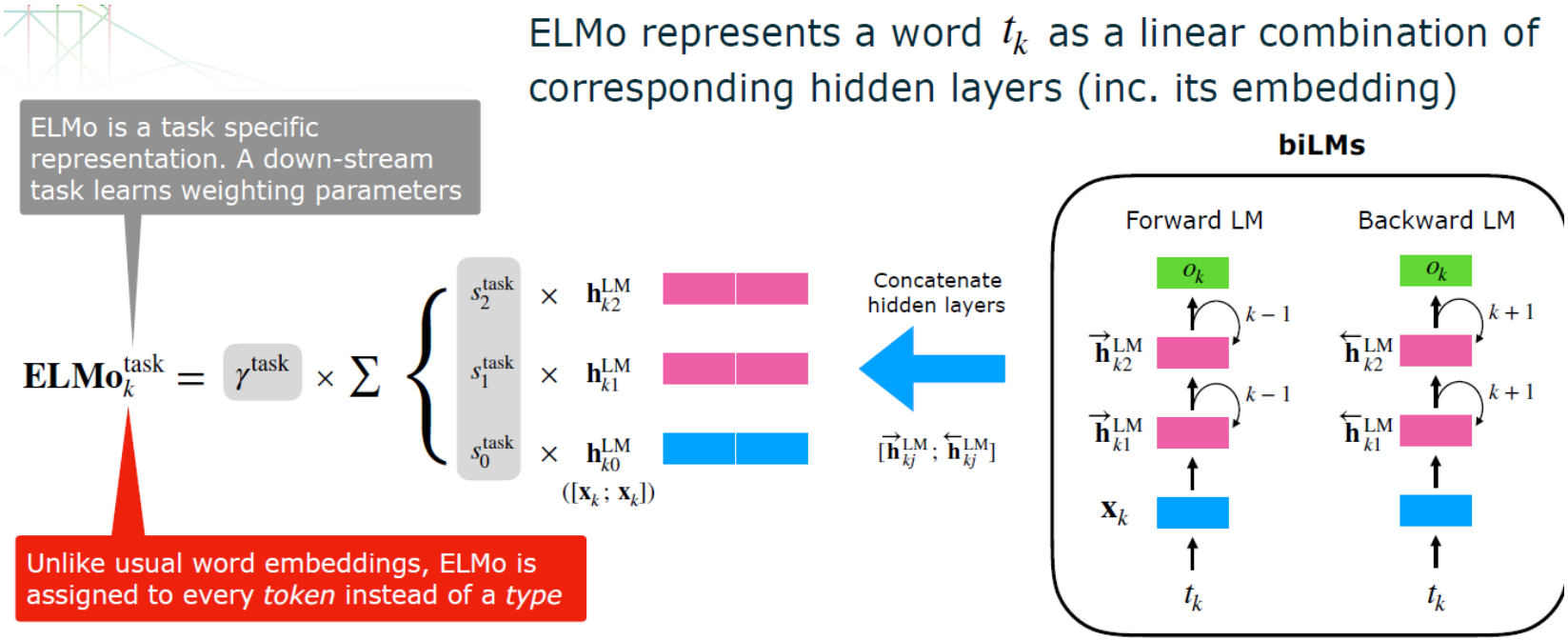
With long short term memory (LSTM) network, predicting the next words in both directions to build biLMs



https://fr2.slideshare.net/shuntaroy/a-review-of-deep-contextualized-word-representations-peters-2018

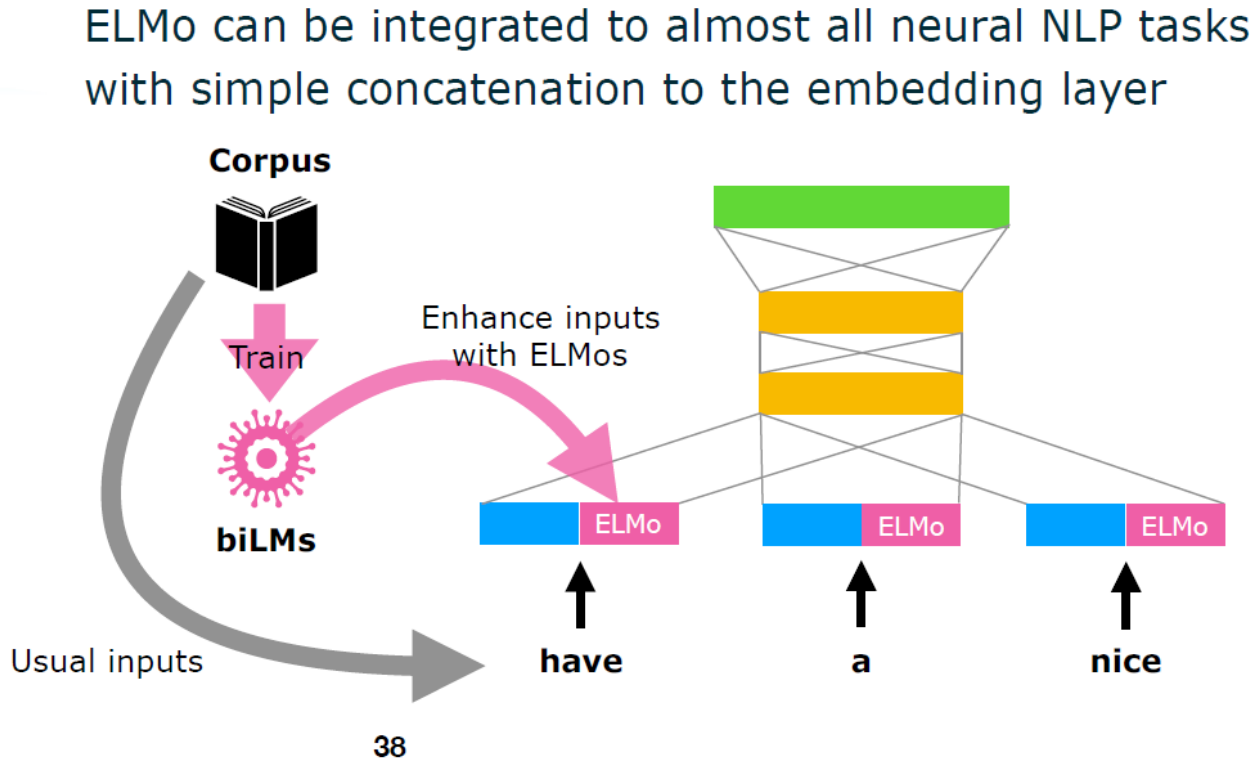
Large size language models based on transformers
ELMo

- The item representation is context dependent



https://fr2.slideshare.net/shuntaroy/a-review-of-deep-contextualized-word-representations-peters-2018

Large size language models based on transformers
ELMo



https://fr2.slideshare.net/shuntaroy/a-review-of-deep-contextualized-word-representations-peters-2018

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

- GPT (Radford et al. 2018), GPT 2 (Radford et al. 2019), GPT 3 (Radford et al. 2020) 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 (Large Language Models)

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

- The decoder model
  - Basically a masked – autoregressive model
  - More details in http://jalammar.github.io/illustrated-gpt2/
- Open AI Blog on GPT2
  - https://openai.com/blog/better-language-models/
  - Paper
    - https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf
- GPT3
  - Paper
    - https://arxiv.org/pdf/2005.14165.pdf
  - API released in 2020
    - https://openai.com/blog/openai-api/
  - Demos
    - https://beta.openai.com/
    - https://beta.openai.com/examples/

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

- Downstream tasks beyond language modeling
  - GPT (Radford et al. 2018)
    - Classification, Entailment, Similarity, Q/A with multiple choices

Downstream tasks (fine tuning)

Transformer model
Unsupervised training

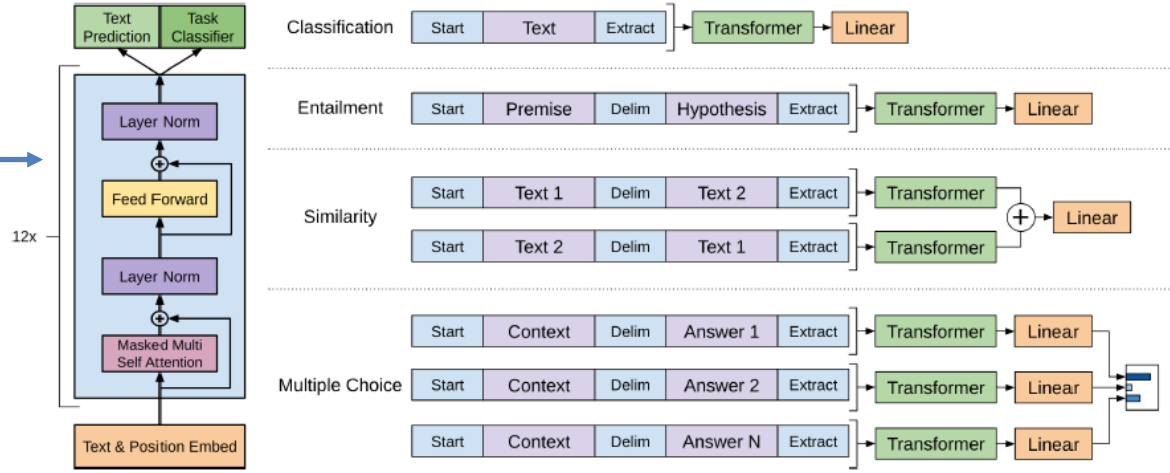


Figure 1: (**left**) Transformer architecture and training objectives used in this work. (**right**) Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.

  - Context slot for the downstream tasks: for Q/A (multiple choices) contains text + questions

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

- Test tasks (not trained on)
  - Language modeling on test datasets it has not been trained on – possibly different from the web training dataset
  - Predict the final word of sentences
  - Reading comprehension
    - Conditioning: document, associated conversation (sequence of questions and answers about the text, final question GPT is asked to answer)
  - Summarization
  - Translation
    - Conditioning
      - Sequence of example pairs of the format english sentence = french sentence, and a final english sentence =
      - Greedy decoding is then used on the output of GPT2, first generated sentence is used as translation
  - Question answering

Large size language models based on transformers
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
  - Starts the exploration of LLM for solving a variety of language tasks
  - At the core of further edvelopments like ChatGPT

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

- Importance of size

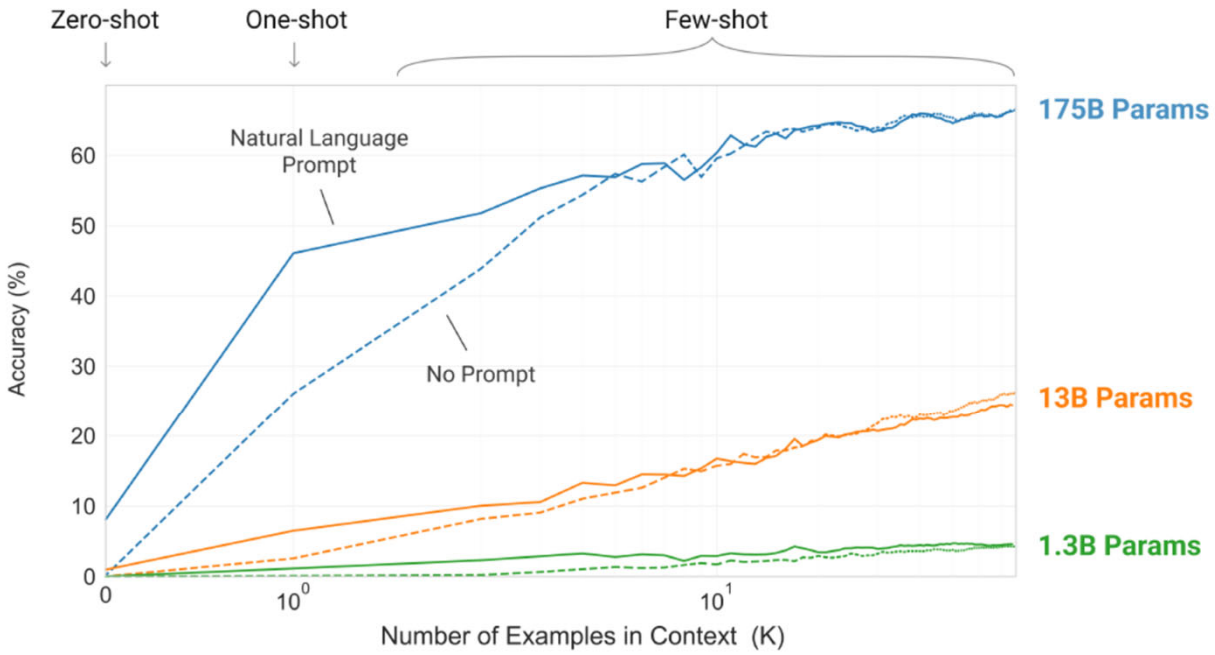


Figure 1.2: Larger models make increasingly efficient use of in-context information. We show in-context learning performance on a simple task requiring the model to remove random symbols from a word, both with and without a natural language task description (see Sec. 3.9.2). The steeper "in-context learning 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
GPT family (OpenAI) – GPT3

- Few shot etc

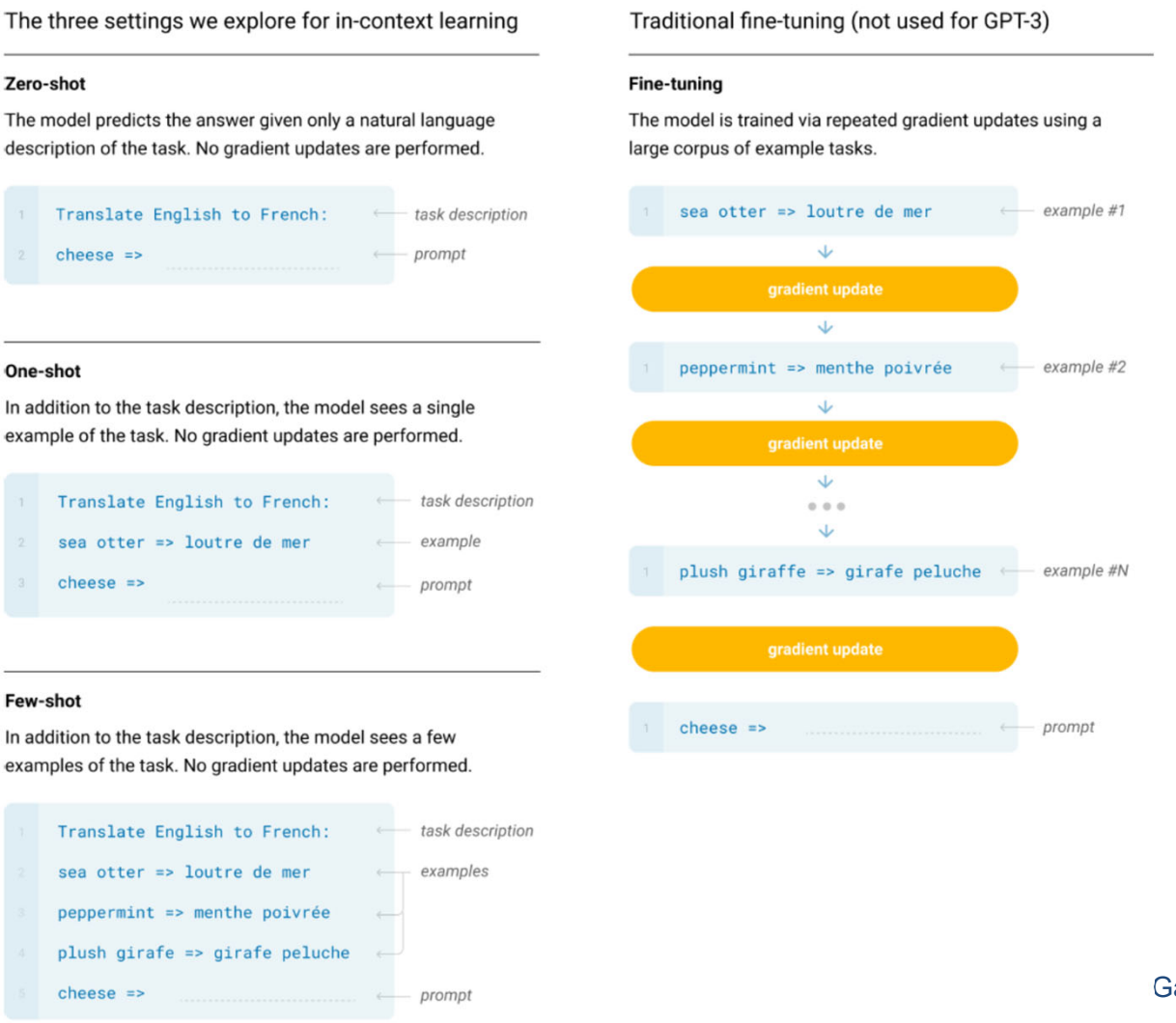


Figure 2.1: Zero-shot, one-shot and few-shot, contrasted with traditional fine-tuning. The panels above show four methods for performing a task with a language model – fine-tuning is the traditional method, whereas zero-, one-, and few-shot, which we study in this work, require the model to perform the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

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

- Arithmetic
  - To test GPT-3's ability to perform simple arithmetic operations without task-specific training, we developed a small battery of 10 tests that involve asking GPT-3 a simple arithmetic problem in natural language:
  - • 2 digit addition (2D+) – The model is asked to add two integers sampled uniformly from [0; 100), phrased in the form of a question, e.g. "Q: What is 48 plus 76? A: 124."
  - • 2 digit subtraction (2D-) – The model is asked to subtract two integers sampled uniformly from [0; 100); the answer may be negative. Example: "Q: What is 34 minus 53? A: -19".
  - • 3 digit addition (3D+) – Same as 2 digit addition, except numbers are uniformly sampled from [0; 1000).

Context →	Q: What is (2 * 4) * 6? A:
Target Completion →	48

Figure G.42: Formatted dataset example for Arithmetic 1DC

Context →	Q: What is 17 minus 14? A:
Target Completion →	3

Figure G.43: Formatted dataset example for Arithmetic 2D-

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

- See prompting and few shot examples starting p 50 on https://arxiv.org/pdf/2005.14165.pdf
- Few shot translation
  - Training dataset contains 93% english words and 7% non english
  - Language model trained on this corpus (no translation training)
  - Evaluated on aligned datasets not seen during training

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

Context →	Analysis of instar distributions of larval I. verticalis collected from a series of ponds also indicated that males were in more advanced instars than females. =
Target Completion →	L'analyse de la distribution de fréquence des stades larvaires d'I. verticalis dans une série d'étangs a également démontré que les larves mâles étaient à des stades plus avancés que les larves femelles.

Figure G.38: Formatted dataset example for En→Fr

Context →	Adevărul este că vă doriţi, cu orice preţ şi împotriva dorinţei europenilor, să continuaţi negocierile de aderare a Turciei la Uniunea Europeană, în ciuda refuzului continuu al Turciei de a recunoaşte Ciprul şi în ciuda faptului că reformele democratice au ajuns într-un punct mort. =
Target Completion →	The truth is that you want, at any price, and against the wishes of the peoples of Europe, to continue the negotiations for Turkey's accession to the European Union, despite Turkey's continuing refusal to recognise Cyprus and despite the fact that the democratic reforms are at a standstill.

Figure G.41: Formatted dataset example for Ro→En

- Choosing an answer
  - PIQA
    - Common sense questions on the physical world
  - COPA
    - A task from the superGLUE dataset

Context →	How to apply sealant to wood.
Correct Answer →	Using a brush, brush on sealant onto wood until it is fully saturated with the sealant.
Incorrect Answer →	Using a brush, drip on sealant onto wood until it is fully saturated with the sealant.

Figure G.4: Formatted dataset example for PIQA

Context →	My body cast a shadow over the grass because
Correct Answer →	the sun was rising.
Incorrect Answer →	the grass was cut.

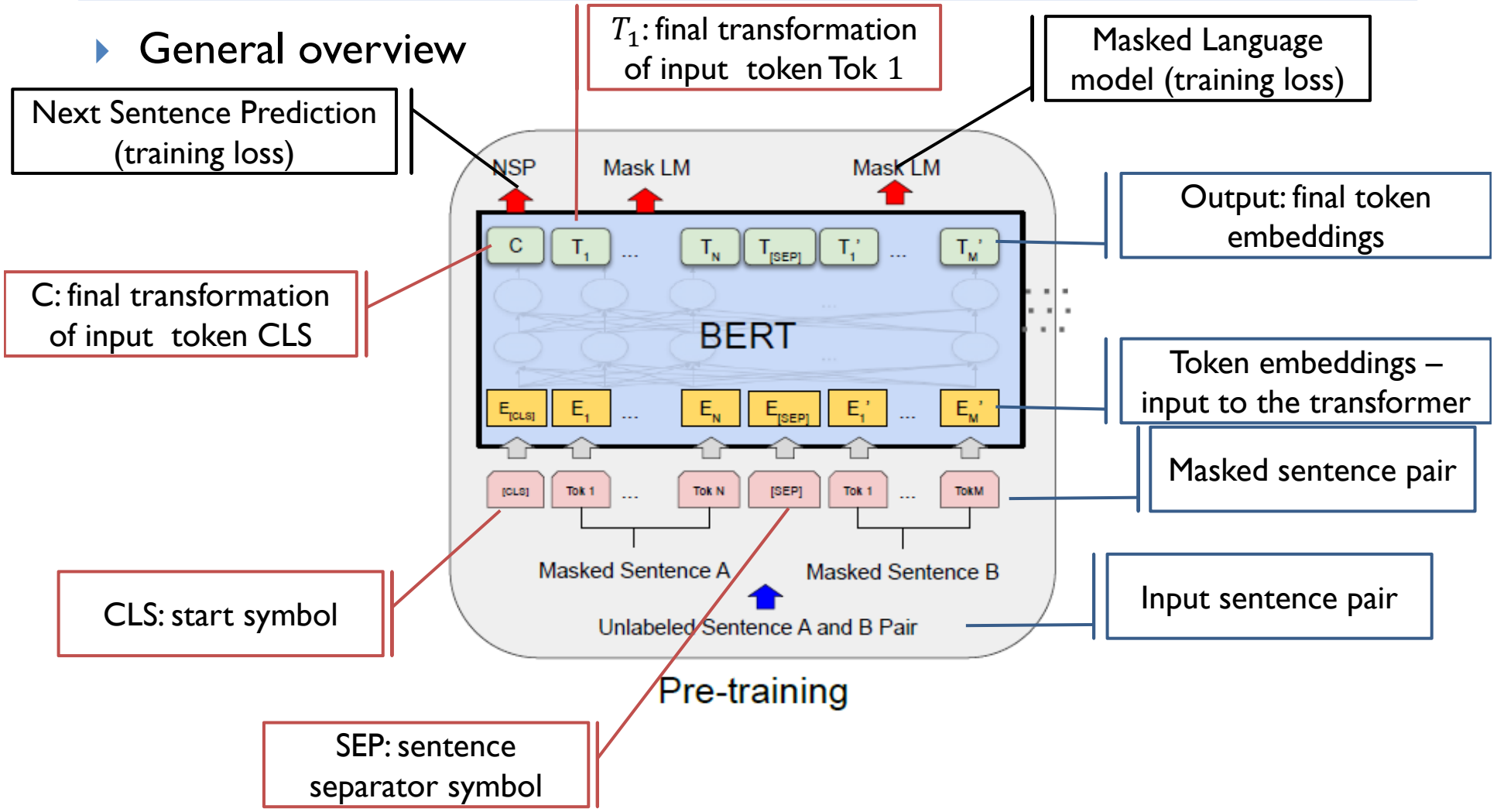
Figure G.5: Formatted dataset example for COPA

Large size language models based on transformers
BERT family (Google)

- BERT family is a reference transformer model family
  - BERT: Bidirectional Encoder Representations from Transformers
  - It comes in many variants, see e.g. the available implementations in the Hugging Face library, https://huggingface.co/
  - It is used in many different contexts
    - e.g. multilingual BERT (about 100 languages)
- As with GPT, BERT proceeds in two steps
  - Unsupervised language model training on large corpora
  - Supervised fine tuning for a variety of tasks
- Originality
  - Two training criteria
    - Masked Language Model (MLM) + Next Sentence Prediction (NSP)
    - Remember: downstream tasks may be at the token (MLM criterion) or sequence (NSP criterion) level
  - Bidirectional Encoder: considers a whole sequence at each step and not only past information like in auto-regressive models (GPT)
  - The same architecture is used for unsupervised training and fine tuning (except from output layers specific to downstream tasks)

Large size language models based on transformers
BERT family (Google)

- General overview



Large size language models based on transformers
BERT family (Google)

- Input representation

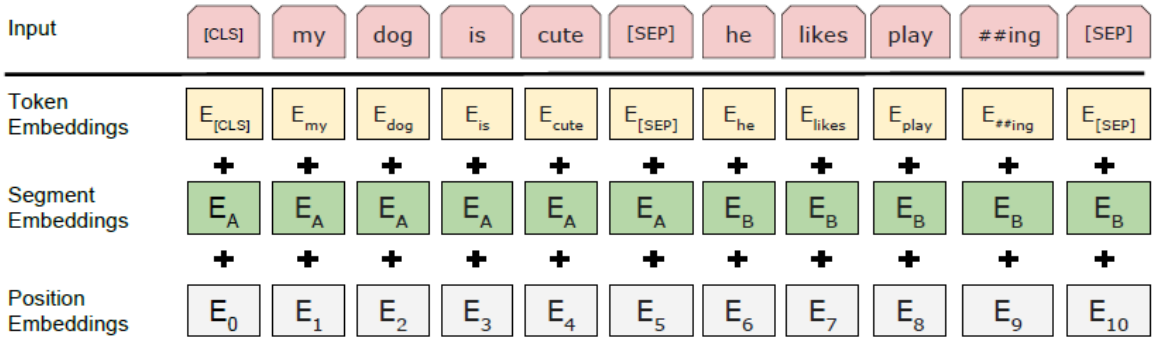
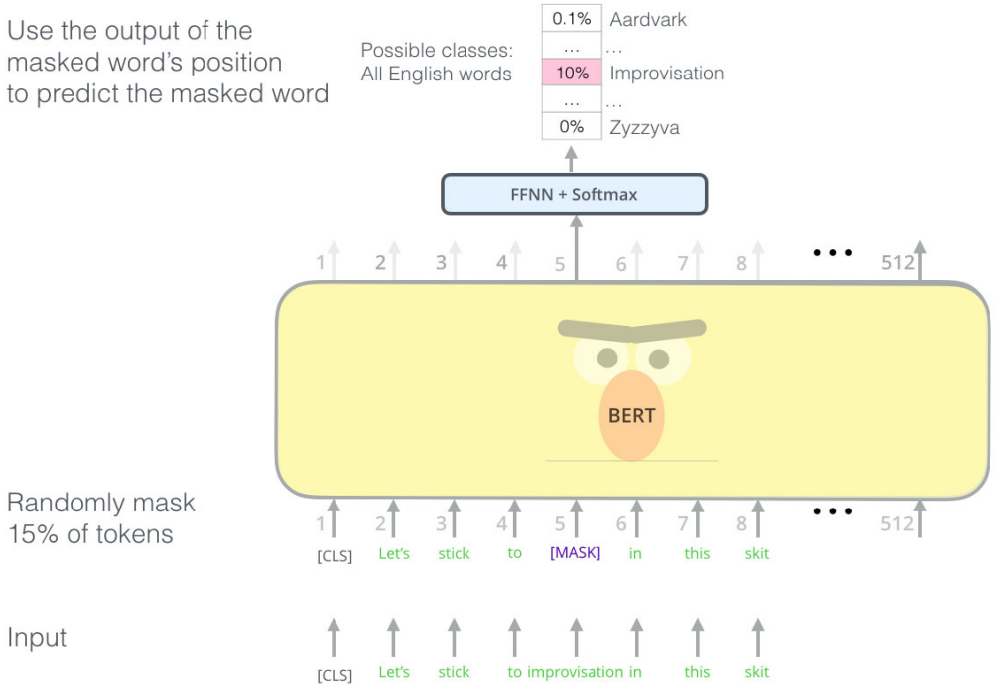


Figure 2: BERT input representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings.

- The initial token is always the special symbol CLS
  - The final hidden state corresponding to this token is used as the input sequence agregate representation for classification tasks
  - Embeddings: **WordPiece** Embeddings with a 30k token vocabulary (detailed later)
- Segment embedding indicates 1st or 2nd sentence (learned)
- Position embeddings
  - As in the transformer description or relative position depending on the model

Large size language models based on transformers
BERT family (Google)

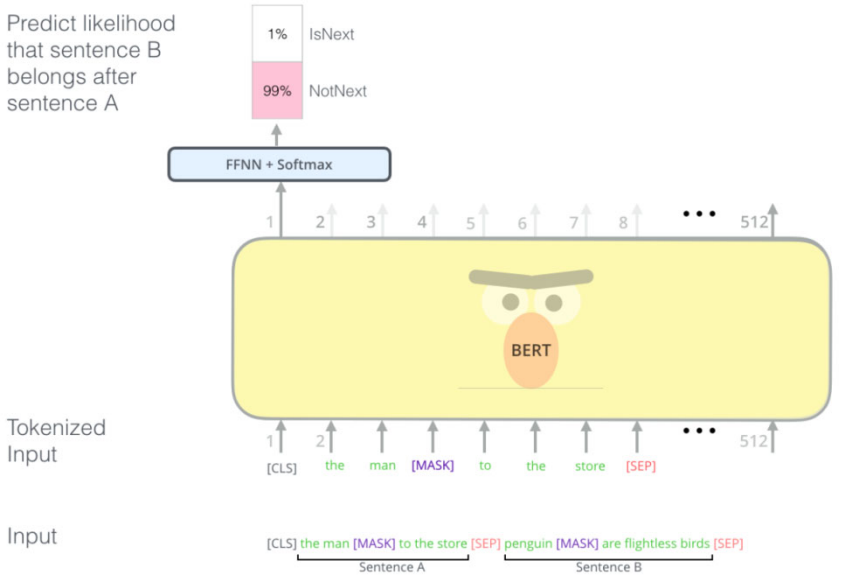
- Training criteria
  - Masked Language Model - MLM
    - Mask 15% of the input tokens at random and predict the masked tokens.
    - The final hidden vector corresponding to the Masked token are fed to a softmax layer as in classical Language Models
      - Note: additional tricks are used in practice for the masking



BERT's clever language modeling task masks 15% of words in the input and asks the model to predict the missing word.

Large size language models based on transformers
BERT family (Google)

- Training criteria
  - Next Sentence Prediction - NSP
    - 2 classes classification problem: is sentence B following sentence A in the corpus?
      - Training on 50% positive/ negative samples
      - 1^{st} item output
      - This is supposed to encode whole input sentences



The second task BERT is pre-trained on is a two-sentence classification task. The tokenization is oversimplified in this graphic as BERT actually uses WordPieces as tokens rather than words --- so some words are broken down into smaller chunks.

Large size language models based on transformers
BERT family (Google)

- Pre-training data
  - Books Corpus (800 M words)
  - English Wikipedia (2500 M words)

Large size language models based on transformers
BERT family (Google)

- Fine tuning
  - Plug the task specific inputs and outputs into BERT and fine tune end to end.
  - At the output, the token representations are fed into an output layer for token level tasks (sequence Tagging like NER, Q/A) and the CLS representation is fed into an output layer for classification (e.g. entailment, sentiment analysis)

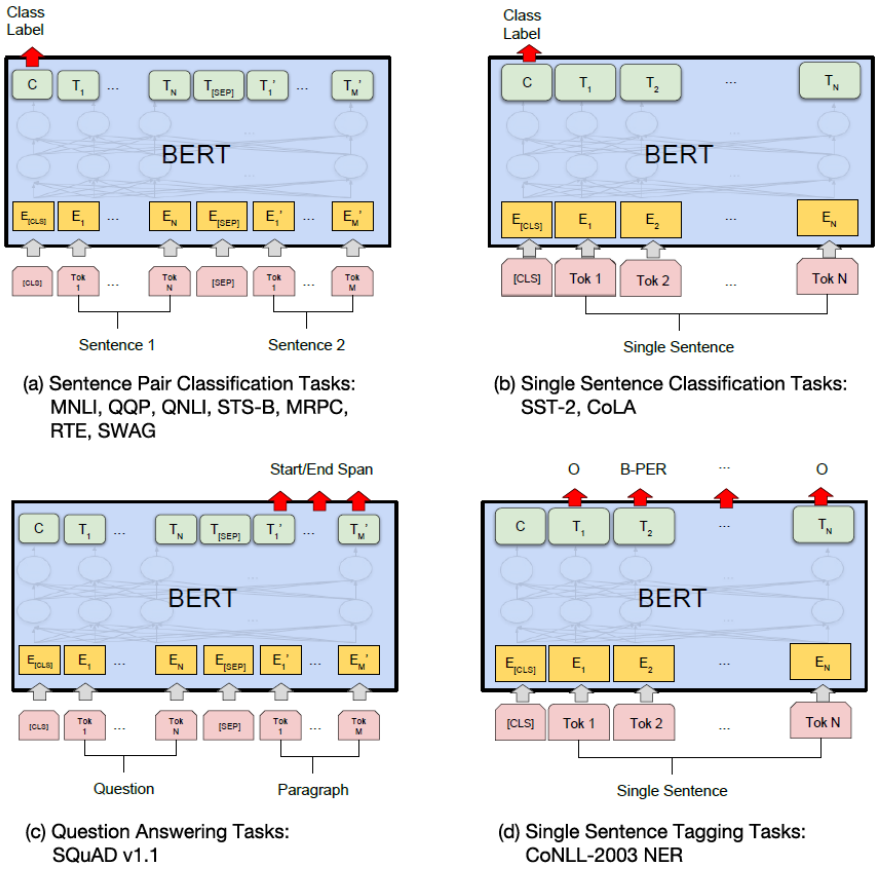
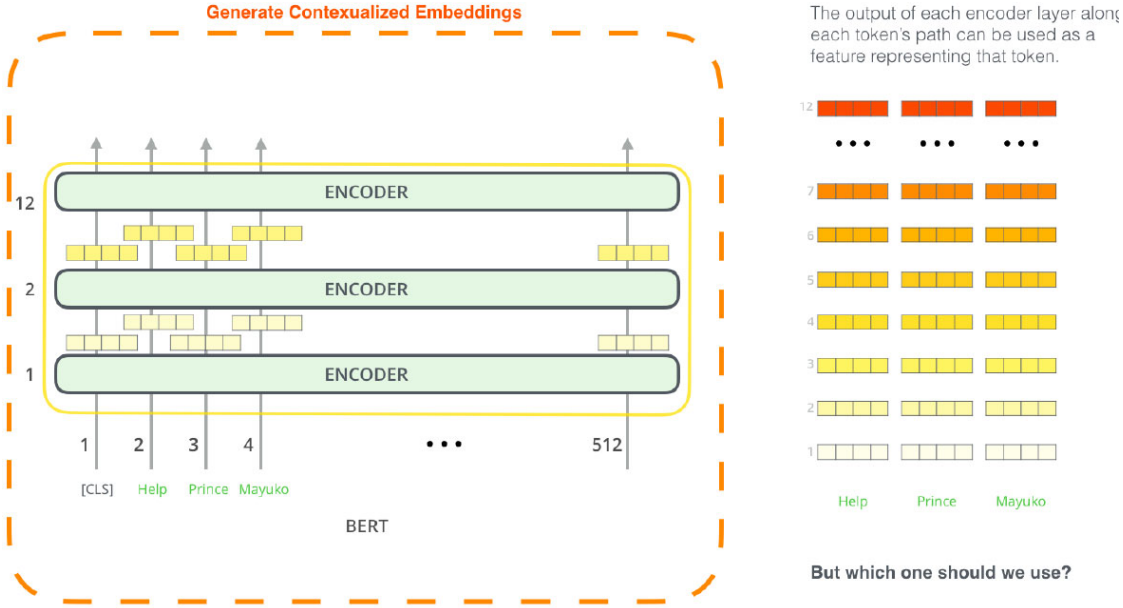


Figure 4: Illustrations of Fine-tuning BERT on Different Tasks.

Large size language models based on transformers
BERT family (Google)

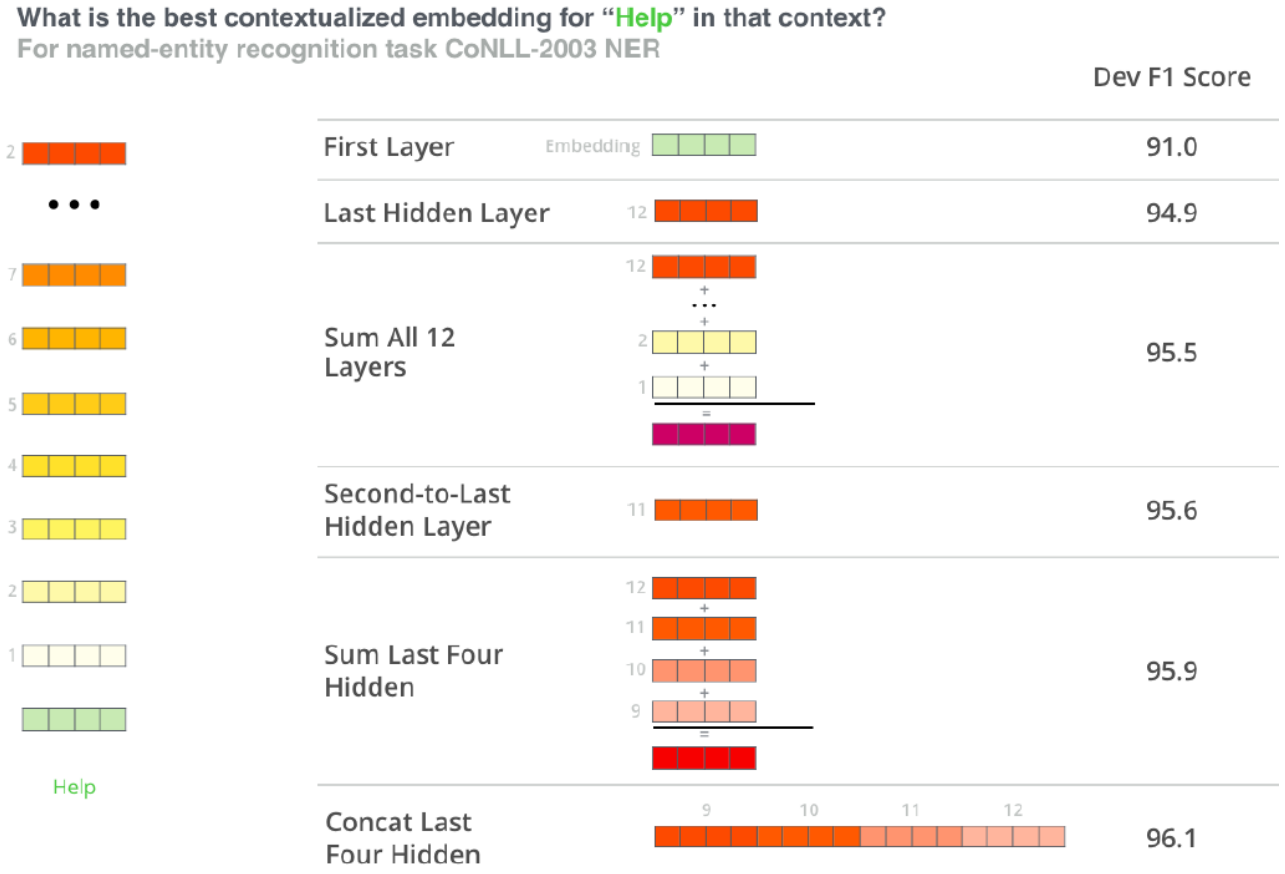
- Feature Learning
  - Instead of fine tuning, the model could be used to extract token representations from a pre-train model. The token are then fed into task specific architectures without fine tuning of the token representations (as with Word2Vec).
    - The paper indicates performance not far from fine tuning



Which vector works best as a contextualized embedding? I would think it depends on the task. The paper examines six choices (Compared to the fine-tuned model which achieved a score of 96.4):

Large size language models based on transformers
BERT family (Google)

- Feature Learning



Large size language models based on transformers
BERT family (Google)

- RoBERTa (Liu et al 2019)
  - Follow up of BERT, analyzes key hyperparameters of BERT and proposes efficient strategies
  - Has became a reference for BERT like architectures
  - Main findings
    - MLM training criterion is enough, no need for NSP
    - Training with large batches improves performance
    - More training data improves performance

Large size language models based on transformers
T5 (Google)

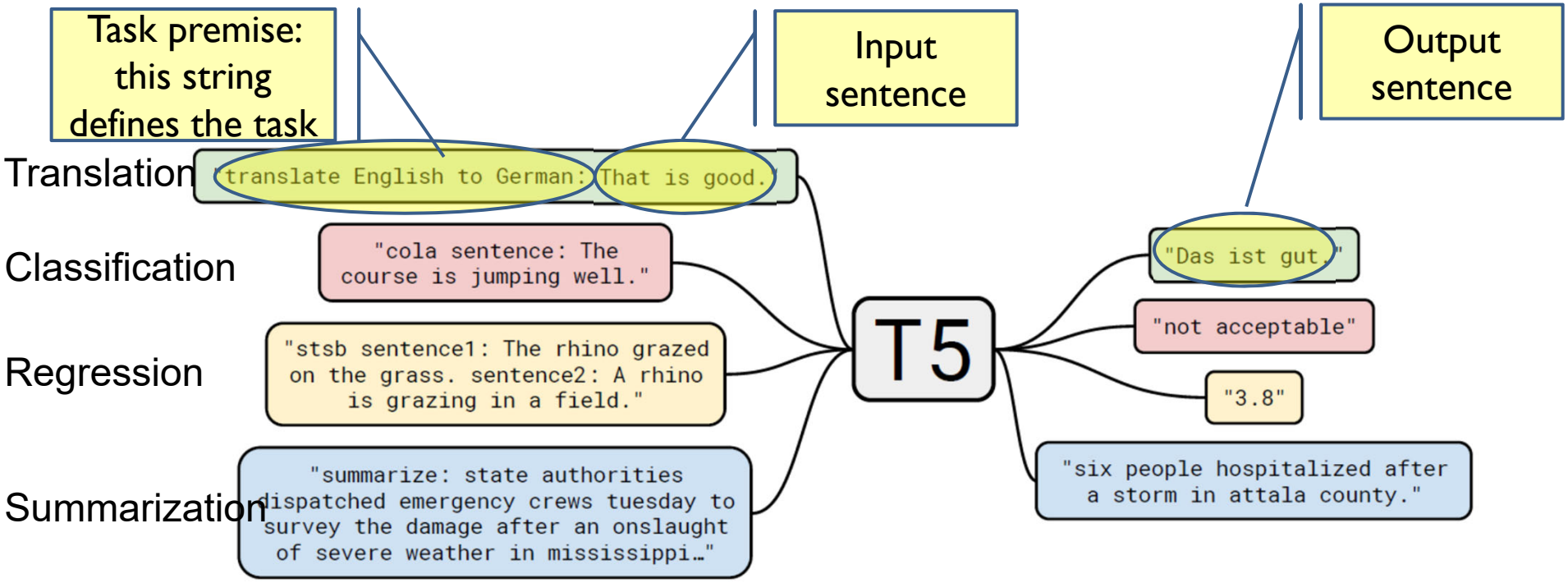
- Illustrations from
  - Raffel, C., et al. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. JMLR. 21, (2020), 1–67.
  - Slides: https://colinraffel.com/talks/mila2020transfer.pdf

- Objective of the paper
  - Explore different strategies for large size Tranformers on a variety of NLP tasks
    - model architectures, pre-training and fine tuning training objectives, transfer learning, scaling, etc

- Strategy
  - Introduce a Text-to-Text framework allowing handling several NLP tasks in a unified way

Large size language models based on transformers
T5 (Google)

- Framework: Text-to-Text Transfer Transformer (T5)
  - Reformulate NLP tasks used in classical benchmarks (classification, summarization, translation) in a Text-to-Text framework
  - Both input and output are textual strings
    - Evaluate within this unified framework different model design choices

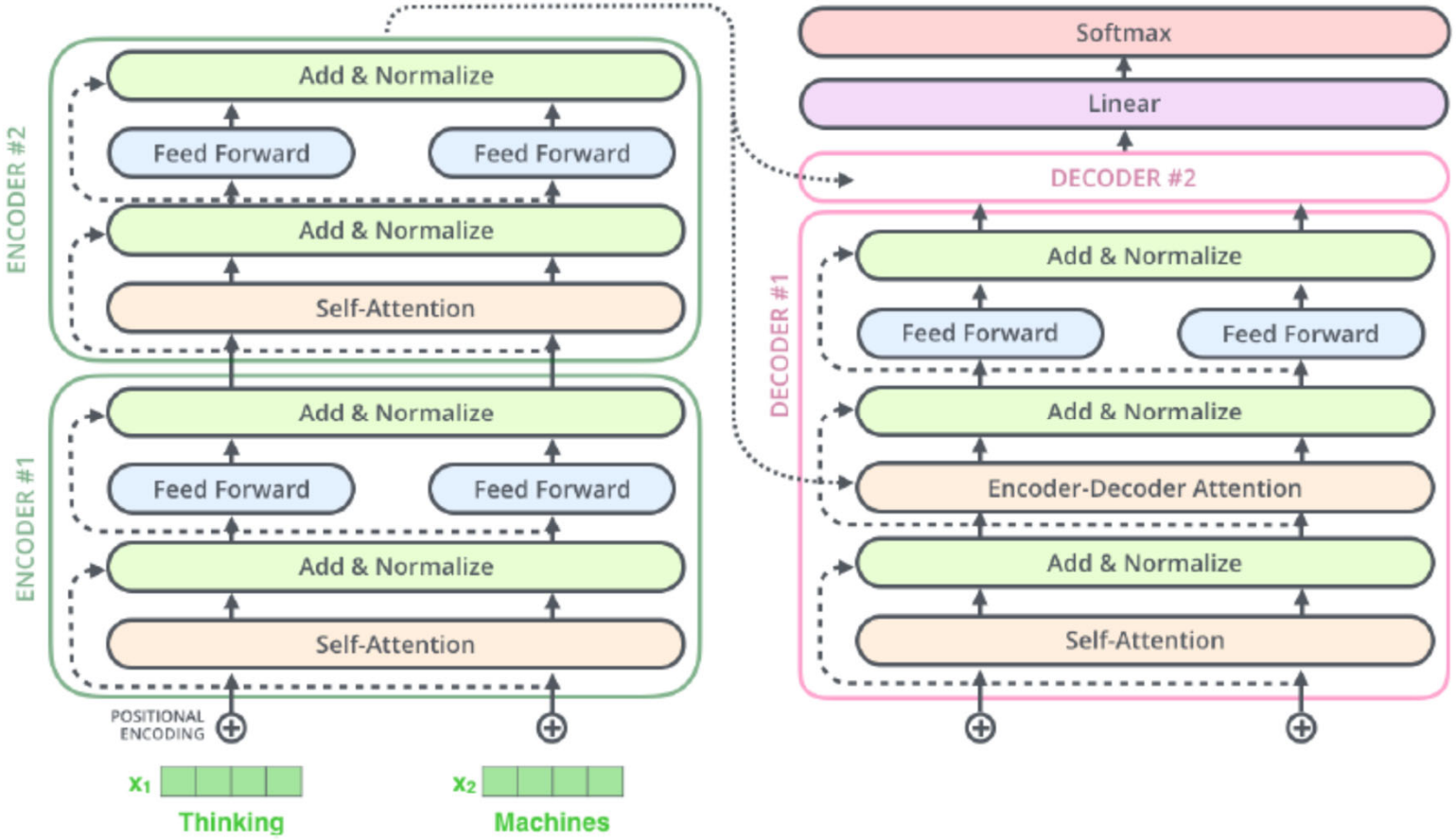


Large size language models based on transformers
T5 (Google)

- Framework: Text-to-Text Transfer Transformer (T5)
  - Text-to-Text requires a decoder to generate text
    - BERT encoders are designed to produce a single output per token, i.e. they are ok for classification tasks or text span selection, not directly applicable for generation
  - This frameworks allows them to use maximum likelihood (typically cross-entropy) as a training objective for both pretraining and fine tuning
  - Note:
    - at test time, they use greedy decoding
    - Vocabulary: Sentencepiece with a 32 k vocabulary
- Examples how to reframe NLP tasks in T5
  - Translation
    - Input: « translate English to German: That is good », translate English to German is a premise (a promt) that defines the task
    - Output: « das ist gut »
  - Text classification
    - MNLI benchmark: goal is to predict wether a premise implies (« entailment »), contradicts (« contradiction ») or neither (« neutral ») a hypothesis
    - Input: « mnli premise: I hate pigeons. Hypothesis: my feeling towards pigeons are filled with animosity »
    - Output: target word « entailment »

Large size language models based on transformers
T5 (Google) - illustration: J. Alammar 2018

- T5 architecture:
  - different choices, best one is Encoder + Decoder close to the original Transformer (Vaswani 2017)



Large size language models based on transformers
T5 (Google)

- Pre-training dataset 750 GB of text extracted from the web and cleaned (below examples of the cleaning process)
  - Available at https://www.tensorflow.org/datasets/catalog/c4

Common Crawl Web Extracted Text

Menu

Lemon

Introduction

The lemon, Citrus Limon (l.) Osbeck, is a species of small evergreen tree in the flowering plant family rutaceae.
The tree's ellipsoidal yellow fruit is used for culinary and non-culinary purposes throughout the world, primarily for its juice, which has both culinary and cleaning uses. The juice of the lemon is about 5% to 6% citric acid, with a ph of around 2.2, giving it a sour taste.

Article

The origin of the lemon is unknown, though lemons are thought to have first grown in Assam (a region in northeast India), northern Burma or China.
A genomic study of the lemon indicated it was a hybrid between bitter orange (sour orange) and citron.

Please enable JavaScript to use our site.

Home
Products
Shipping
Contact
FAQ

Dried Lemons, $3.59/pound

Organic dried lemons from our farm in California.
Lemons are harvested and sun-dried for maximum flavor.
Good in soups and on popcorn.

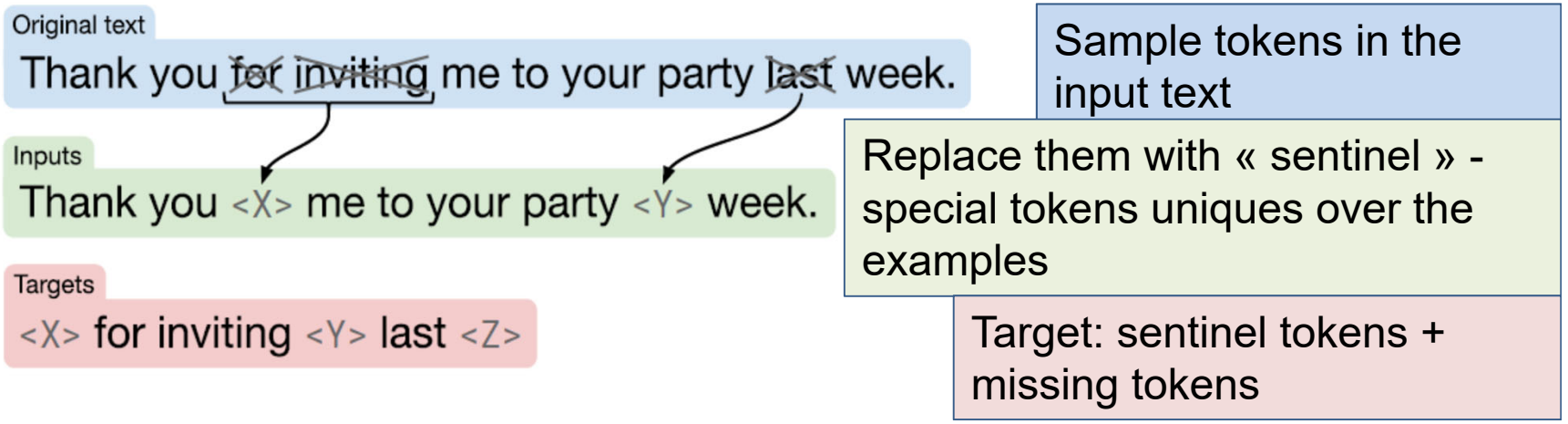
The lemon, Citrus Limon (l.) Osbeck, is a species of small evergreen tree in the flowering plant family rutaceae.
The tree's ellipsoidal yellow fruit is used for culinary and non-culinary purposes throughout the world, primarily for its juice, which has both culinary and cleaning uses. The juice of the lemon is about 5% to 6% citric acid, with a ph of around 2.2, giving it a sour taste.

Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Curabitur in tempus quam. In mollis et ante at consectetur.
Aliquam erat volutpat.
Donec at lacinia est.
Duis semper, magna tempor interdum suscipit, ante elit molestie urna, eget efficitur risus nunc ac elit.
Fusce quis blandit lectus.
Mauris at mauris a turpis tristique lacinia at nec ante.
Aenean in scelerisque tellus, a efficitur ipsum.
Integer justo enim, ornare vitae sem non, mollis fermentum lectus.
Mauris ultrices nisl at libero porta sodales in ac orci.

```
function Ball(r) {
  this.radius = r;
  this.area = pi * r ** 2;
  this.show = function(){
    drawCircle(r);
  }
}
```

Large size language models based on transformers
T5 (Google)

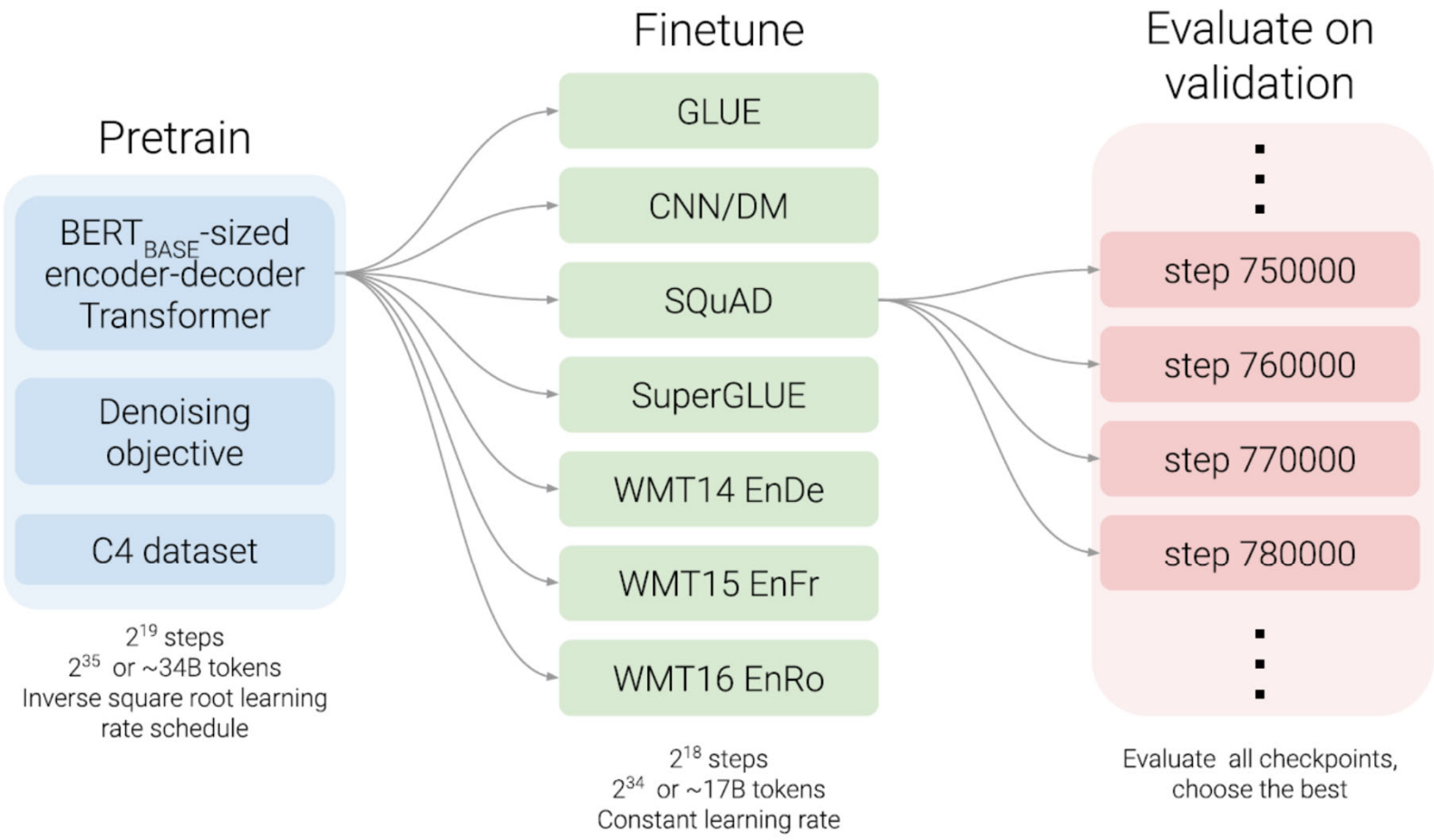
- Unsupervised training objective
  - Best one is similar to MLM in BERT (other choices discussed later)



Schematic of the objective we use in our baseline model. In this example, we process the sentence "Thank you for inviting me to your party last week." The words "for", "inviting" and "last" (marked with an ×) are randomly chosen for corruption. Each consecutive span of corrupted tokens is replaced by a sentinel token (shown as <X> and <Y>) that is unique over the example. Since "for" and 'inviting" occur consecutively, they are replaced by a single sentinel <X>. The output sequence then consists of the dropped-out spans, delimited by the sentinel tokens used to replace them in the input plus a final sentinel token <Z>.

Large size language models based on transformers
T5 (Google)

- Workflow

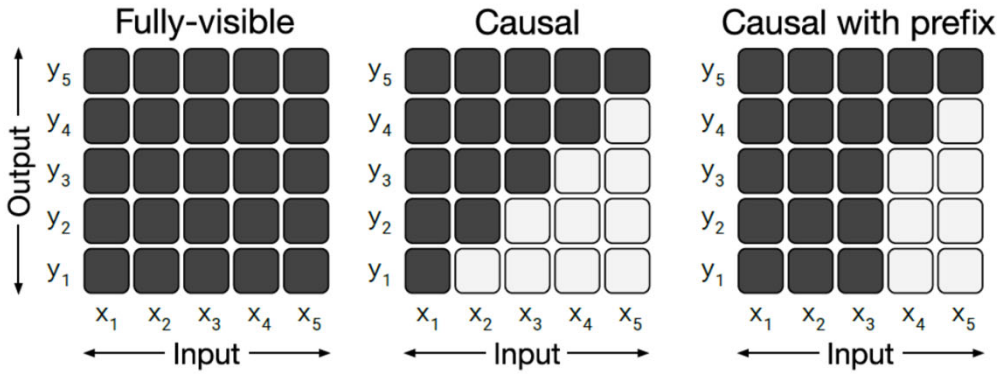


Large size language models based on transformers
T5 (Google)

- Large scale comparison
  - Comparing different hyperparameters, like architecture, training criteria, multitask versus pretraining + fine tuning, etc.
- Main findings
  - Text-to-Text provides a simple way to train a single model on a variety of tasks
  - Original encoder-decoder scheme works best in the T2T framework
  - Objective: the MLM objective is superior to classical language based prediction
  - Transfer training: fine tuning the whole model works better than tuning task specific modules only
  - Scale: larger models, more data increase the performance

Large size language models based on transformers
T5 (Google)

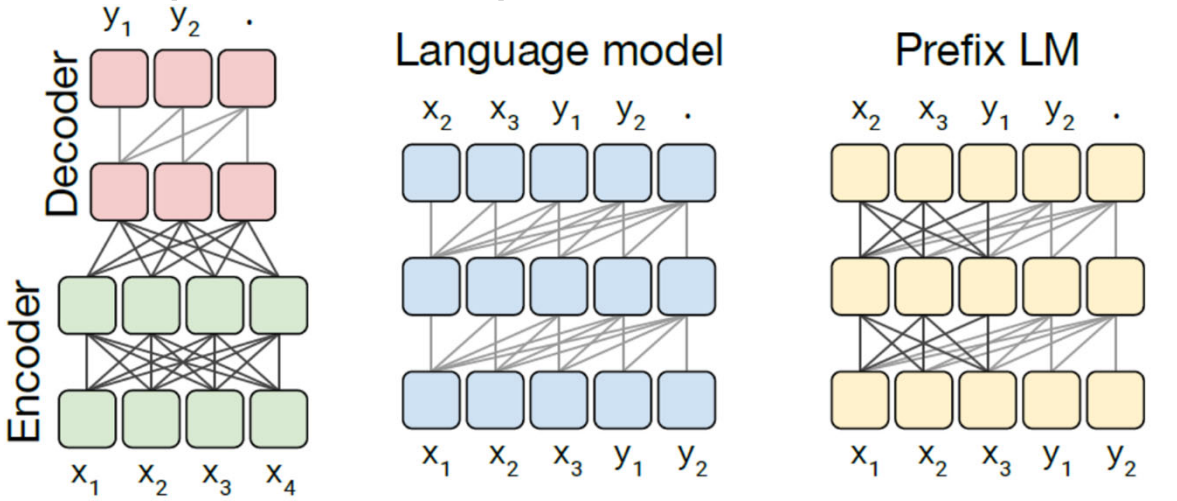
- Large scale comparison, example: Architectures evaluated
  - 3 types of architectures involving 3 attention patterns
    - Fully-visible: similar to BERT
    - Causal: similar to GPT
    - Causal with prefix: allows full attention of part of the input



Matrices representing different attention mask patterns. The input and output of the self-attention mechanism are denoted x and y respectively. A dark cell at row i and column j indicates that the self-attention mechanism is allowed to attend to input element j at output timestep i. A light cell indicates that the self-attention mechanism is *not* allowed to attend to the corresponding i and j combination. Left: A fully-visible mask allows the self-attention mechanism to attend to the full input at every output timestep. Middle: A causal mask prevents the ith output element from depending on any input elements from "the future". Right: Causal masking with a prefix allows the self-attention mechanism to use fully-visible masking on a portion of the input sequence.

Large size language models based on transformers
T5 (Google)

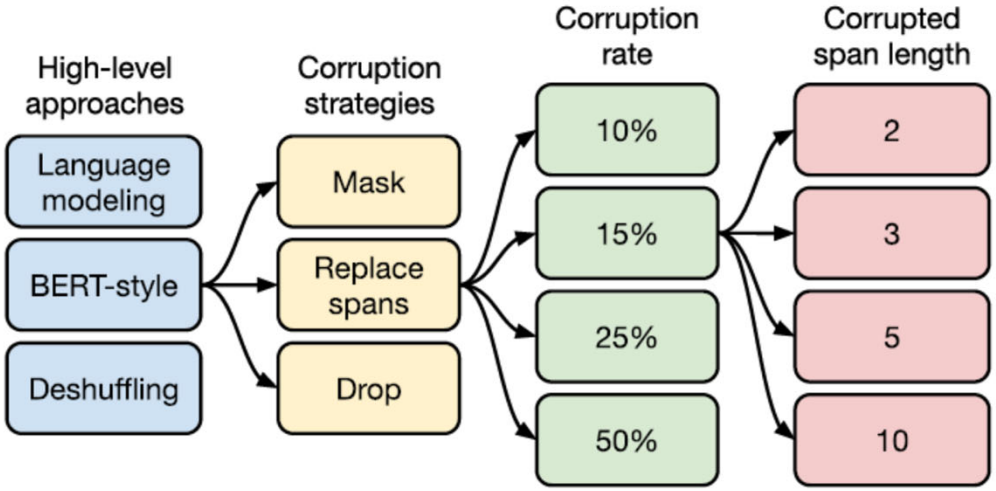
- Large scale comparison, example: 3 architectures evaluated



Schematics of the Transformer architecture variants we consider. In this diagram, blocks represent elements of a sequence and lines represent attention visibility. Different colored groups of blocks indicate different Transformer layer stacks. Dark grey lines correspond to fully-visible masking and light grey lines correspond to causal masking. We use "." to denote a special end-of-sequence token that represents the end of a prediction. The input and output sequences are represented as x and y respectively. Left: A standard encoder-decoder architecture uses fully-visible masking in the encoder and the encoder-decoder attention, with causal masking in the decoder. Middle: A language model consists of a single Transformer layer stack and is fed the concatenation of the input and target, using a causal mask throughout. Right: Adding a prefix to a language model corresponds to allowing fully-visible masking over the input.

Large size language models based on transformers
T5 (Google)

- Large scale comparison, example: different objectives for training



A flow chart of our exploration of unsupervised objectives. We first consider a few disparate approaches in Section 3.3.1 and find that a BERT-style denoising objective performs best. Then, we consider various methods for simplifying the BERT objective so that it produces shorter target sequences in Section 3.3.2. Given that replacing dropped-out spans with sentinel tokens performs well and results in short target sequences, in Section 3.3.3 we experiment with different corruption rates. Finally, we evaluate an objective that intentionally corrupts contiguous spans of tokens in Section 3.3.4.

Large size language models based on transformers
T5 (Google)

- Summary of experiments

Encoder-decoder architecture

Architecture	Params	Cost	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Encoder-decoder	$2P$	M	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	**39.82**	**27.65**
Enc-dec, shared	P	M	82.81	18.78	**80.63**	**70.73**	26.72	39.03	**27.46**
Enc-dec, 6 layers	P	$M/2$	80.88	18.97	77.59	68.42	26.38	38.40	26.95
Language model	P	M	74.70	17.93	61.14	55.02	25.09	35.28	25.86
Prefix LM	P	M	81.82	18.61	78.94	68.11	26.43	37.98	27.39

Span prediction objective

Span length	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Baseline (i.i.d.)	**83.28**	19.24	80.88	71.36	**26.98**	**39.82**	**27.65**
2	**83.54**	19.39	**82.09**	**72.20**	**26.76**	**39.99**	**27.63**
3	**83.49**	**19.62**	**81.84**	**72.53**	**26.86**	39.65	**27.62**
5	**83.40**	19.24	**82.05**	**72.23**	**26.88**	39.40	**27.53**
10	82.85	19.33	**81.84**	70.44	**26.79**	39.49	**27.69**

C4 dataset

Dataset	Size	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ C4	745GB	83.28	**19.24**	80.88	71.36	**26.98**	**39.82**	**27.65**
C4, unfiltered	6.1TB	81.46	19.14	78.78	68.04	26.55	39.34	27.21
RealNews-like	35GB	**83.83**	**19.23**	80.39	72.38	**26.75**	**39.90**	**27.48**
WebText-like	17GB	**84.03**	**19.31**	**81.42**	71.40	**26.80**	**39.74**	**27.59**
Wikipedia	16GB	81.85	**19.31**	81.29	68.01	**26.94**	39.69	**27.67**
Wikipedia + TBC	20GB	83.65	**19.28**	**82.08**	**73.24**	**26.77**	39.63	**27.57**

Multi-task pre-training

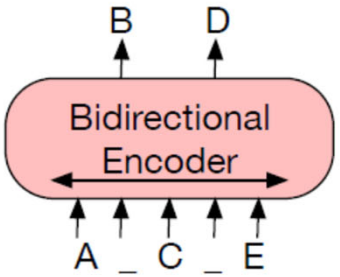
Training strategy	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
★ Unsupervised pre-training + fine-tuning	**83.28**	**19.24**	**80.88**	**71.36**	**26.98**	39.82	27.65
Multi-task training	81.42	**19.24**	79.78	67.30	25.21	36.30	27.76
Multi-task pre-training + fine-tuning	**83.11**	**19.12**	**80.26**	**71.03**	**27.08**	39.80	**28.07**
Leave-one-out multi-task training	81.98	19.05	79.97	**71.68**	**26.93**	39.79	**27.87**
Supervised multi-task pre-training	79.93	18.96	77.38	65.36	26.81	**40.13**	**28.04**

Bigger models trained longer

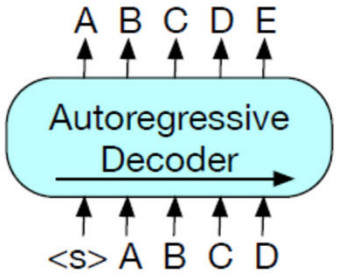
Scaling strategy	GLUE	CNNDM	SQuAD	SGLUE	EnDe	EnFr	EnRo
Baseline	83.28	19.24	80.88	71.36	26.98	39.82	27.65
1× size, 4× training steps	85.33	19.33	82.45	74.72	27.08	40.66	27.93
1× size, 4× batch size	84.60	19.42	82.52	74.64	27.07	40.60	27.84
2× size, 2× training steps	**86.18**	19.66	**84.18**	77.18	27.52	**41.03**	28.19
4× size, 1× training steps	**85.91**	19.73	**83.86**	**78.04**	27.47	40.71	28.10
4× ensembled	84.77	**20.10**	83.09	71.74	**28.05**	40.53	**28.57**
4× ensembled, fine-tune only	84.05	19.57	82.36	71.55	27.55	40.22	28.09

Large size language models based on transformers

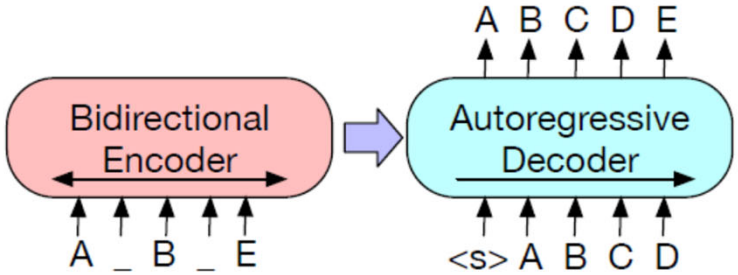
- Recap on models architectures
  - Different schemes for using Transformers (figure from Lewis, et al. 2019. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension).



(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.

Tokenization

Tokenization

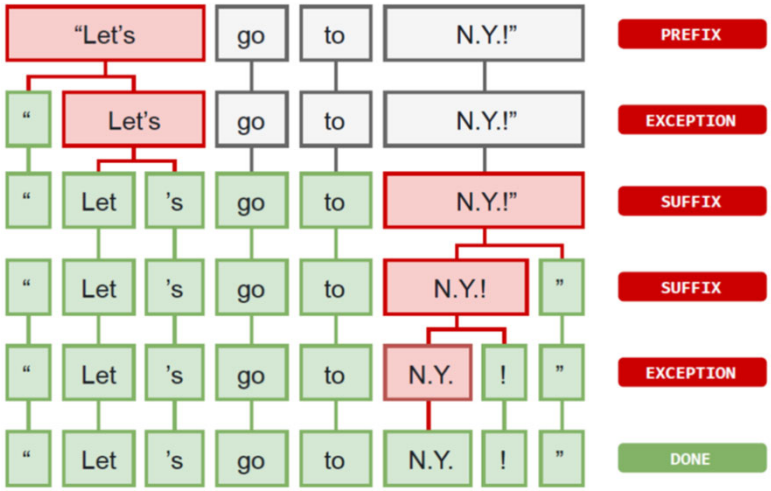
- A text is a sequence of characters
- An important step is the segmentation of the sequence into meaningful units – this is calleds tokenization
  - All the methods for dealing with NLP (RNNs, Transformers) use some form of tokenization.
  - **This means that a pretrained model should be used with the corresponding tokenization**
- Note
  - This is not the only one preprocessing step, cleaning, e.g. lowercase, or other normalization operations might be performed.

Tokenization

- Example from:
  - https://huggingface.co/transformers/tokenizer_summary.html
- Consider the sentence:
  - "Don't you love Transformers? We sure do."
- Naive tokenization methods
  - Split words by spaces
    - ["Don't", "you", "love", "Transformers?", "We", "sure", "do."]
  - Split items by spaces and punctuation
  - ["Don", "'", "t", "you", "love", "Transformers", "?", "We", "sure", "do", "."]

Tokenization

- Rule based tokenizers
  - spaCy: a **free, open-source library** for NLP in Python. It offers a rule based tokenizer. spaCY splits on spaces and then looks individual substrings: looks for special tokens (may be user defined), and splits off prefixes, suffixes, infixes.
  - Results in (too) large vocabulary – not used with transformers



  - For the sentence "Don't you love Transformers? We sure do." this would give (https://spacy.io/usage/spacy-101#annotations)
    - ["Do", "n't", "you", "love", "Transformers", "?", "We", "sure", "do", "."]

Tokenization - Subword tokenization - examples

```
>>> from transformers import BertTokenizer

>>> tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")

>>> tokenizer.tokenize("I have a new GPU!")
["i", "have", "a", "new", "gp", "##u", "!"]
```

```
>>> from transformers import XLNetTokenizer

>>> tokenizer = XLNetTokenizer.from_pretrained("xlnet-base-cased")

>>> tokenizer.tokenize("Don't you love Transformers? We sure do.")
["▁Don", "'", "t", "▁you", "▁love", "▁", "Transform", "ers", "?", "▁We", "▁sure", "▁do", "."]
```

Tokenization -Subword tokenization
Byte-pair encoding (Sennrich et al. 2015)

- Relies on a pre-tokenizer that splits training data into words
  - e.g. space tokenization, spaCy, etc
  - Then compute the frequency of each word
- Algorithm
  - Split all words into unicode characters – this constitutes the initial **vocabulary**
  - While the vocabulary limit size is not reached
    - Find the most frequent symbol bigram in the vocabulary
    - Merge the symbols to create a new symbol and **add this new symbol to the vocabulary**
  - Size of vocabulary and # merge operations are parameters of the algorithm
  - Used in GPT (478 base symbols and 40 k merges)
  - GPT2 uses a variant, replacing unicode characters by **Bytes** and using 256 bytes as base symbols (a unigram character may need multiple bytes for its encoding) and 50 k merges plus an « unk » symbol for symbols not seen during training, i.e. a 50257 dictionary size
    - With Byte BPE, no need for « unk » symbol, all the Bytes are seen during training

Tokenization -Subword tokenization
Byte-pair encoding (Sennrich et al. 2015)

- Example

Dictionary (5 words)				Frequency
h	u	g		10
p	u	g		5
p	u	n		12
b	u	n		4
h	u	g	s	5

Vocabulary (7 symbols)
b, g, h, n, p, s, u

Pair (u,g) is the most frequent (20) bigram, add a new symbol, « ug » in the vocabulary, and merge the corresponding representations

Dictionary				Frequency
h	ug			10
p	ug			5
p	u	n		12
b	u	n		4
h	ug	s		5

Vocabulary
b, g, h, n, p, s, u, ug

Pair (u,n) is the most frequent (16) bigram, add a new symbol, « un » in the vocabulary, and merge the corresponding representations

Tokenization -Subword tokenization
Byte-pair encoding (Sennrich et al. 2015)

- Example

Dictionary				Frequency
h	ug			10
p	ug			5
p	un			12
b	un			4
h	ug	s		5

Vocabulary
b, g, h, n, p, s, u, ug, un

Pair (h, « ug ») is the most frequent (15) bigram, add a new symbol, « ug » in the vocabulary, and merge the corresponding representations

Dictionary				Frequency
hug				10
p	ug			5
p	un			12
b	un			4
hug	s			5

Vocabulary
b, g, h, n, p, s, u, ug, un, hug

At test time, all the new text is decomposed according to the final dictionary, e.g. « bug » is tokenized as [« b », »ug »] and symbols not seen during training are replaced by a special symbol « unk »

Tokenization -Subword tokenization
Byte-pair encoding (Sennrich et al. 2015)

- Merge is performed at the word level and not at the level of whole sentences or sequences
  - This is to save computation cost
    - If there are N symbols, naive implementation of most frequent bigram requires $O(N^2)$ operations

Tokenization - Subword tokenization
Wordpiece (Schuster 2012) – BERT uses a variant of Wordpiece

- Similar to BPE, but merge rule changes
- Instead of merging the most frequent bigrams, Wordpiece merges the symbol pair that maximises the likelihood of a unigram language model trained on the training data, once added to the vocabulary
- Log likelihood at step t
  - $L\big(Vocabulary(t)\big) = \sum_{x_i \in Vocabulary(t)} \log p(x_i)$
- If we fusion symbols x_j and x_k, the new log likelihod is
  - $L\big(Vocabulary(t+1)\big) = L\big(Vocabulary(t)\big) + \log \frac{p(x_j, x_k)}{p(x_j)p(x_k)}$
- Then one merges the couple x_j and x_k that maximizes $\log \frac{p(x_j, x_k)}{p(x_j)p(x_k)}$

  This is the mutual information between the 2 symbols

Tokenization - Subword tokenization
Sentencepiece (Kudo 2018) – used in XLNet

- Does not use pre-tokenization but considers the text as a raw input stream then including space and separation characters.
- Makes use of BPE or Unigram (another tokenizer not described here) for constructing the appropriate vocabulary.
  - Makes use of a special data structure (priority queue based algorithm) to reduce the asymptotic runtime from $O(N^2)$ to $O(NlogN)$
- Properties
  - Could be used easily on a variety of languages including languages that do not use spaces to separate words (e.g. Chinese)
  - Does not require any language specific tokenizers

Dowsnstream tasks used to evaluate large transformers models

- Classification tasks – GLUE and Super Glue Benchmarks
  - MNLI Multi-Genre Natural Language Inference
    - is a large-scale, crowdsourced entailment classification task (Williams et al., 2018). Given a pair of sentences, the goal is to predict whether the second sentence is an entailment, contradiction, or neutral with respect to the first one.
  - QQP Quora Question Pairs
    - is a binary classification task where the goal is to determine if two questions asked on Quora are semantically equivalent (Chen et al., 2018).
  - QNLI Question Natural Language Inference
    - Is a version of the Stanford Question Answering Dataset (Rajpurkar et al., 2016) which has been converted to a binary classification task (Wang et al., 2018a). The positive examples are (question, sentence) pairs which do contain the correct answer, and the negative examples are (question, sentence) from the same paragraph which do not contain the answer.
  - SST-2 The Stanford Sentiment Treebank
    - is a binary single-sentence classification task consisting of sentences extracted from movie reviews with human annotations of their sentiment (Socher et al., 2013).

Dowsnstream tasks used to evaluate large transformers models

- CoLA The Corpus of Linguistic Acceptability
  - is a binary single-sentence classification task, where the goal is to predict whether an English sentence is linguistically "acceptable" or not (Warstadt et al., 2018).
- STS-B The Semantic Textual Similarity Benchmark
  - is a collection of sentence pairs drawn from news headlines and other sources (Cer et al., 2017). They were annotated with a score from 1 to 5 denoting how similar the two sentences are in terms of semantic meaning.
- MRPC Microsoft Research Paraphrase Corpus
  - consists of sentence pairs automatically extracted from online news sources, with human annotations for whether the sentences in the pair are semantically equivalent (Dolan and Brockett, 2005).
- RTE Recognizing Textual Entailment
  - is a binary entailment task similar to MNLI, but with much less training data (Bentivogli et al., 2009).14

Dowsnstream tasks used to evaluate large transformers models

- Question Answering
  - The Stanford Question Answering Dataset (SQuAD v1.1) is a collection of 100k crowdsourced question/answer pairs (Rajpurkar et al., 2016). Given a question and a passage from Wikipedia containing the answer, the task is to predict the answer text span in the passage.
  - The SQuAD 2.0 task extends the SQuAD 1.1 problem definition by allowing for the possibility that no short answer exists in the provided paragraph, making the problem more realistic.
- Q/A with multiple choices
  - The Situations With Adversarial Generations (SWAG) dataset contains 113k sentence-pair completion examples that evaluate grounded commonsense inference (Zellers et al., 2018). Given a sentence, the task is to choose the most plausible continuation among four choices.

References: papers used as illustrations for the presentation

- Arjovsky, M., Chintala, S., & Bottou, L. (2017). Wasserstein Generative Adversarial Networks. In Proceedings of The 34th International Conference on Machine Learning (pp. 1–32). Zhu, J.-Y., Park, T., Isola, P., & Efros, A. A. (2017). Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. In ICCV (pp. 2223–2232).
- Badrinarayanan, V., Kendall, A., & Cipolla, R. (2017). SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 39(12), 2481–2495.
- Bahdanau, D., Cho, K., & Bengio, Y. (2015). Neural Machine Translation By Jointly Learning To Align and Translate. In Iclr 2015. https://doi.org/10.1146/annurev.neuro.26.041002.131047
- Baydin Atilim Gunes , Barak A. Pearlmutter, Alexey Andreyevich Radul, Automatic differentiation in machine learning: a survey. CoRR abs/1502.05767 (2017)
- Belkin, M., Hsu, D., Ma, S., & Mandal, S. (2019). Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences of the United States of America*, *116*(32), 15849–15854. https://doi.org/10.1073/pnas.1903070116
- Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2016). Enriching Word Vectors with Subword Information. Transactions of the Association for Computational Linguistics, 5, 135-146.
- Cadène R., Thomas Robert, Nicolas Thome, Matthieu Cord:M2CAI Workflow Challenge: Convolutional Neural Networks with Time Smoothing and Hidden Markov Model for Video Frames Classification. CoRR abs/1610.05541 (2016)
- Chen M. Denoyer L., Artieres T. Multi-view Generative Adversarial Networks without supervision, 2017 , https://arxiv.org/abs/1711.00305.
- Chen, L. C., Papandreou, G., Kokkinos, I., Murphy, K., & Yuille, A. L. (2018). DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs. IEEE Transactions on Pattern Analysis and Machine Intelligence, 40(4), 834–848. https://doi.org/10.1109/TPAMI.2017.2699184
- Cho, K., Gulcehre, B. van M.C., Bahdanau, D., Bougares, F., Schwenk, H. and Bengio, Y. 2014. Learning Phrase Representations using RNN Encoder – Decoder for Statistical Machine Translation. EMNLP 2014 (2014), 1724–1734.
- Cybenko, G. (1993). Degree of approximation by superpositions of a sigmoidal function. Approximation Theory and Its Applications, 9(3), 17–28.
- Dumoulin, V., & Visin, F. (2016). A guide to convolution arithmetic for deep learning. In arxiv.org/abs/1603.07285 (pp. 1–31).
- Durand T. , Thome, N. and Cord M., WELDON: Weakly Supervised Learning of Deep Convolutional Neural Networks, CVPR 2016.
- Frome, A., Corrado, G., Shlens, J., Bengio, S., Dean, J., Ranzato, M.A. and Mikolov, T. 2013. DeViSE: A Deep Visual-Semantic Embedding Model. NIPS 2013 (2013).
- Gatys, L. A., Ecker, A. S., & Bethge, M. (2016). Image style transfer using convolutional neural networks. In CVPR (pp. 2414–2423).

References: papers used as illustrations for the presentation

- Goodfellow I, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio , Generative adversarial nets, NIPS 2014, 2672-2680
- Goodfellow, I., Pouget-Abadie, J., & Mirza, M. (2014). Generative Adversarial Networks. NIPS, 2672--2680.
- Guhring et al., 2020, Expressivity of deep neural networks, arXiv:2007.04759
- He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016a. Deep residual learning for image recognition. In CVPR, 770–778.
- He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016b. Identity mappings in deep residual networks. In ECCV, 630–645.
- He, K., Gkioxari, G., Dollar, P., & Girshick, R. (2017). Mask R-CNN. Proceedings of the IEEE International Conference on Computer Vision, 2017–Octob, 2980–2988.
- Hornik, K. (1991). Approximation Capabilities of Multilayer Feedforward Networks [J]. Neural Networks, 4(2), 251–257.
- Ioffe S., Szegedy C.: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, 1995, http://arxiv.org/abs/1502.03167
- Jalammar 2018 - http://jalammar.github.io/illustrated-transformer/
- Johnson, J., Hariharan, B., van der Maaten, L., Fei-Fei, L., Zitnick, C. L., & Girshick, R. (2017). CLEVR: A Diagnostic Dataset for Compositional Language and Elementary Visual Reasoning. In CVPR (pp. 1988–1997). https://doi.org/10.1109/CVPR.2017.215
- Johnson, J., Hariharan, B., van der Maaten, L., Hoffman, J., Fei-Fei, L., Zitnick, C. L., & Girshick, R. (2017). Inferring and Executing Programs for Visual Reasoning. In ICCV (pp. 3008–3017). ttps://doi.org/10.1109/ICCV.2017.325
- Krizhevsky, A., Sutskever, I. and Hinton, G. 2012. Imagenet classification with deep convolutional neural networks. Advances in Neural Information. (2012), 1106–1114.
- Le, Q., Ranzato, M., Monga, R., Devin, M., Chen, K., Corrado, G., Dean, J. and Ng, A. 2012. Building high-level features using large scale unsupervised learning. Proceedings of the 29th International Conference on Machine Learning (ICML-12). (2012), 81–88.
- Lerer, A., Gross, S., & Fergus, R. (2016). Learning Physical Intuition of Block Towers by Example. In Icml (pp. 430–438). Retrieved from http://arxiv.org/abs/1603.01312
- Lin, M., Chen, Q., & Yan, S. (2013). Network In Network. In arxiv.org/abs/1312.4400. https://doi.org/10.1109/ASRU.2015.7404828
- Lin, Z., Feng, M., Santos, C. N. dos, Yu, M., Xiang, B., Zhou, B., & Bengio, Y. (2017). A Structured Self-attentive Sentence Embedding. In *ICLR*.
- Liu, P.J., Saleh, M., Pot, E., Goodrich, B., Sepassi, R., Kaiser, Ł. and Shazeer, N. 2018. Generating wikipedia by summarizing long sequences. ICLR (2018), 1–18.
- Mathieu, M., Couprie, C., & LeCun, Y. (2016). Deep multi-scale video prediction beyond mean square error. In ICLR (pp. 1–14). Retrieved from http://arxiv.org/abs/1511.05440
- Mirza, M., & Osindero, S. (2014). Conditional Generative Adversarial Nets. In arxiv.org/abs/1411.1784.
- Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning. In NIPS Deep Learning Workshop. https://doi.org/10.1038/nature14236
- Nakkiran, P., Kaplum, G., Bansal, Y., Yang, T., Barak, P., & Sutskever, I. (2020). Deep Double Descent: Where Bigger Models and More Data Hurt. *ICLR*, 1–24.
- Pearlmutter B.A., Gradient calculations for dynamic recurrent neural networks: a survey, IEEE Trans on NN, 1995

References: papers used as illustrations for the presentation

- Pennington, J., Socher, R. and Manning, C.D. 2014. GloVe : Global Vectors for Word Representation. EMNLP 2014 (2014), 1532–1543.
- Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. In arxiv.org/abs/1511.06434 (pp. 1–15). https://doi.org/10.1051/0004-6361/201527329
- Radford, Luke Metz, Soumith Chintala, Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, 2016, http://arxiv.org/abs/1511.06434
- Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. In CVPR (pp. 779–788).
- Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B. and Lee, H. 2016. Generative Adversarial Text to Image Synthesis. Icml (2016), 1060–1069.
- Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., & Lee, H. (2016). Generative Adversarial Text to Image Synthesis. In Icml (pp. 1060–1069).
- Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. In MICCAI 2015: Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015 (pp. 234–241).
- Ruder S. (2016). An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747.
- Shelhamer, E., Long, J., Darrell, T., Shelhamer, E., Darrell, T., Long, J., … Darrell, T. (2015). Fully Convolutional Networks for Semantic Segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 3431–3440).
- Srivastava N., Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov: Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research 15(1): 1929-1958 (2014)
- Sutskever, I., Vinyals, O. and Le, Q. V 2014. Sequence to sequence learning with neural networks. Advances in Neural Information Processing Systems (NIPS) (2014), 3104–3112.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., … Polosukhin, I. (2017). Attention Is All You Need. In *NIPS*.
- Vinyals, O., Toshev, A., Bengio, S. and Erhan, D. 2015. Show and Tell: A Neural Image Caption Generator, CVPR 2015: 3156-3164
- Widrow, B., Glover, J. R., McCool, J. M., Kaunitz, J., Williams, C. S., Hearn, R. H., … Goodlin, R. C. (1975). 1975 Adaptive noise cancelling: Principles and applications. Proceedings of the IEEE, 63(12), 1692–1716. https://doi.org/10.1109/PROC.1975.10036
- Wu, Yonghui, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, Jeffrey Dean, Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation, Technical Report, 2016.

References: papers used as illustrations for the presentation

- Xu, K., Ba, J. L., Kiros, R., Cho, K., Courville, A., Salakhutdinov, R., … Bengio, Y. (2015). Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. In Icml-2015 (pp. 2048–2057). https://doi.org/10.1109/72.279181
- Yang, Z., Yang, D., Dyer, C., He, X., Smola, A., & Hovy, E. (2016). Hierarchical Attention Networks for Document Classification. In Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language
- Yu, F., & Koltun, V. (2016). Multi-Scale Context Aggregation by Dilated Convolutions. In *ICLR, arxiv.org/abs/1511.07122*.

Multi-layer Perceptron – SGD Training
Summary of the algorithm with MSE loss + sigmoid units

- The algorithm is described for a MSE loss – similar derivations for other losses
  - MLP with $M+1$ layers of cells numbered 0 (input layer), …, M(output layer), M weight layers numbered $W(1), \ldots, W(M), w_{ij}(m)$ is the weight from cell j in layer $m-1$ to cell i in layer m (and is one of the components of W^m
- Algorithm
  - Sample an example $(\boldsymbol{x}, \boldsymbol{y}), \boldsymbol{x} \in R^n, \mathbf{y} \in R^p$
  - Compute output $\hat{\boldsymbol{y}} = F_W(\boldsymbol{x}), \hat{\boldsymbol{y}} \in R^p$
  - Compute difference $\boldsymbol{\delta} = (\boldsymbol{y} - \hat{\boldsymbol{y}}) = \left(y_1 - \hat{y}_1, \ldots, y_p - \hat{y}_p\right)^T$
  - Back propagate this error from the last weight layer to the first weight layer:
    - $w_{ij}(m) = w_{ij}(m) + \Delta w_{ij}(m) \rightarrow$ update equation for layer m and weight w_{ij}^m
    - $\Delta w_{ij}(m) = \epsilon e_{\mathrm{i}}(m) z_{\mathrm{j}}(m-1) \rightarrow$ gradiant for $w_{ij}(m)$
      - « e » is the quantity that will be back propagated
    - $e_{\mathrm{i}}(M) = \delta_i g'(a_i(M))$ if i is an output cell with $\delta_i = (y_i - \hat{y}_i)$
    - $e_{\mathrm{i}}(m) = g'(a_i(m)) \sum_{h\ parents\ of\ i} e_h(m+1) w_{hi}(m+1)$ if i is not an output cell