COMP5212: Machine Learning

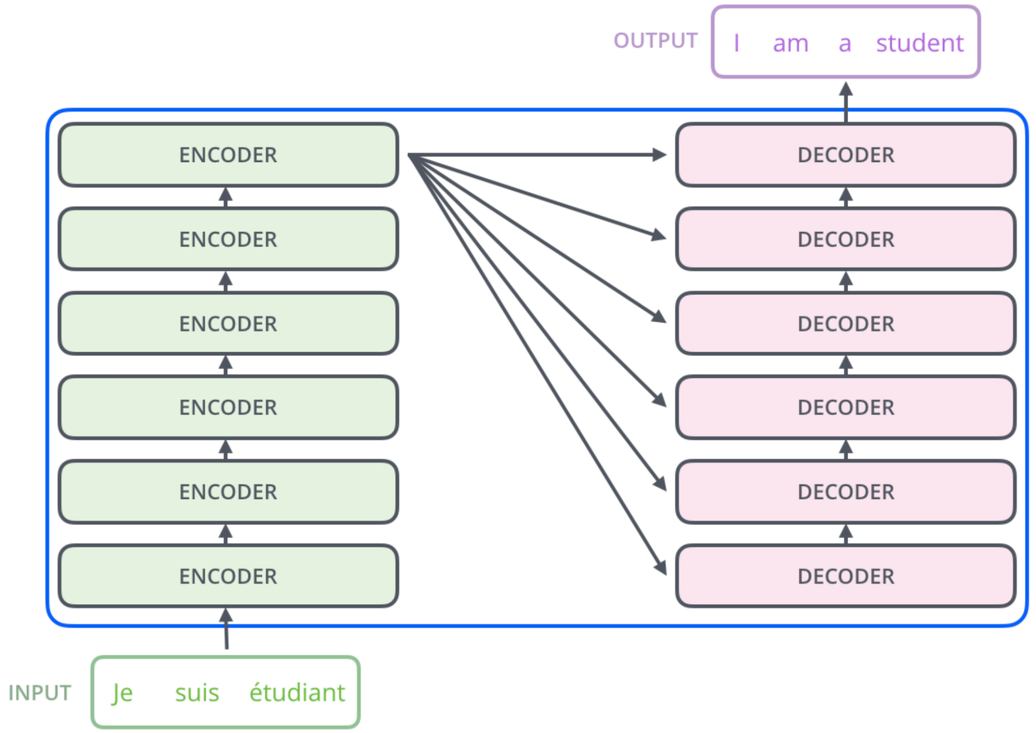
Lecture 16

Minhao Cheng

Transformer

Transformer

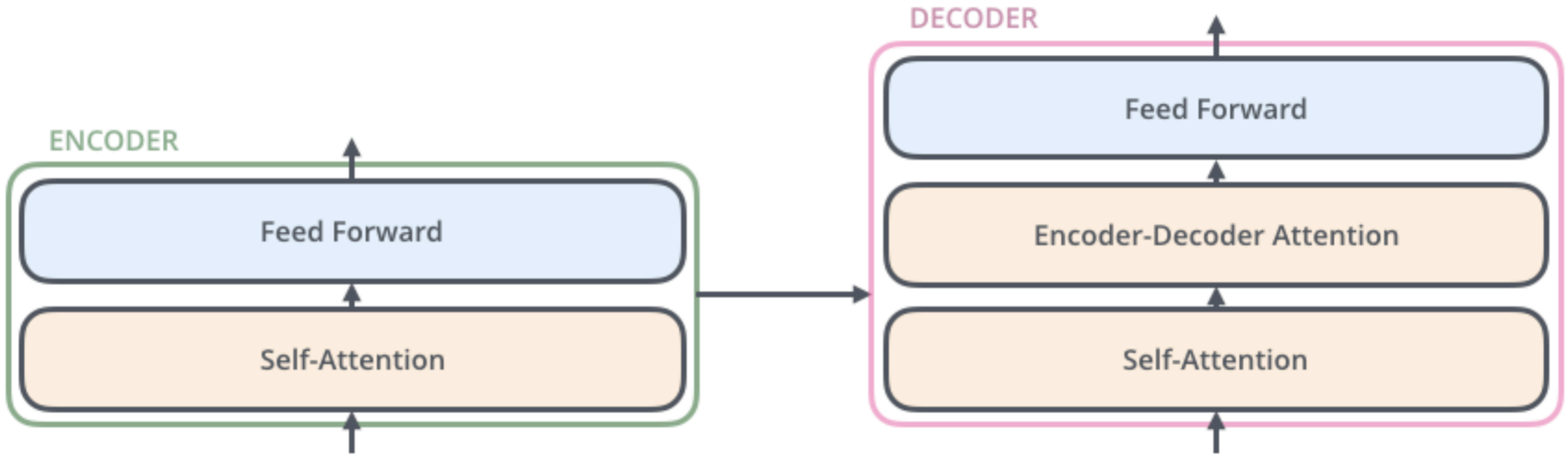
- An architecture that replies entirely on attention without using CNN/RNN
- Proposed in ``Attention Is All You Need'' (Vaswani et al., 2017)
- Initially used for neural machine translation



Transformer

Encoder and Decoder

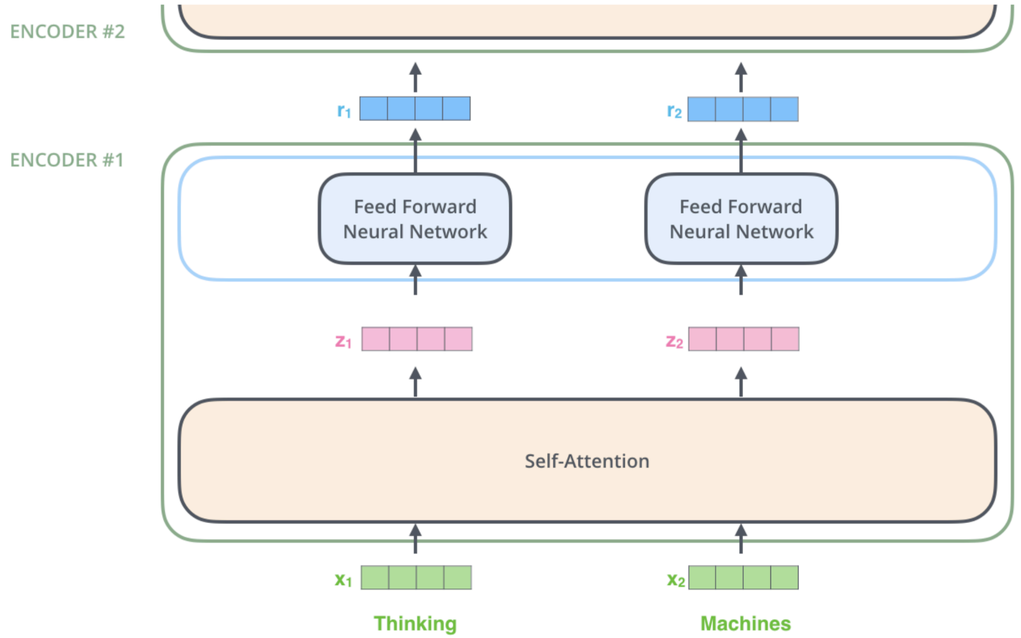
- Self attention layer: the main architecture used in Transformer
- Decoder: will have another attention layer to help it focuses on relevant parts of input sentences.



Transformer

Encoder

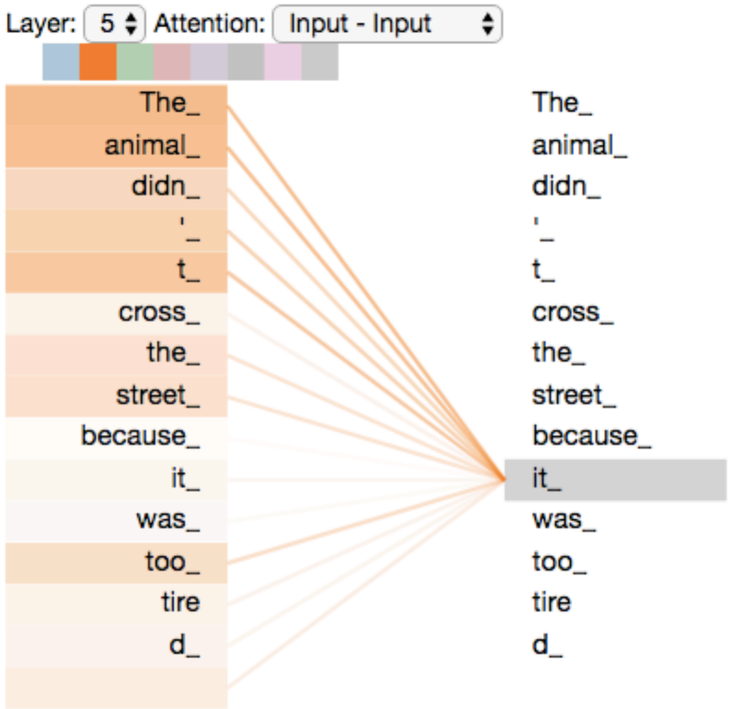
- Each word has a corresponding ``latent vector'' (initially the word embedding for each word)
- Each layer of encoder:
  - Receive a list of vectors as input
  - Passing these vectors to a self-attention layer
  - Then passing them into a feed-foward layer
  - Output a list of vectors



Transformer

Self-attention layer

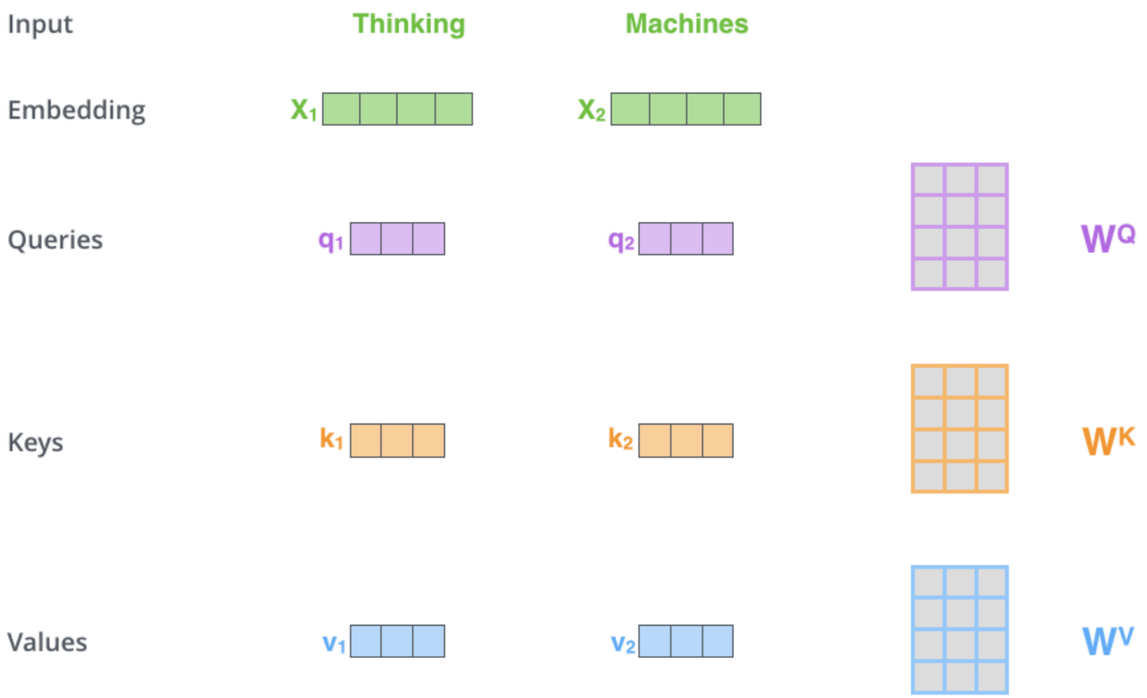
- Main idea: The actual meaning of each word may be related to other words in the sentence
- The actual meaning (latent vector) of each word is a weighted (attention) combination of other words (latent vectors) in the sentences



Transformer

Self-attention layer

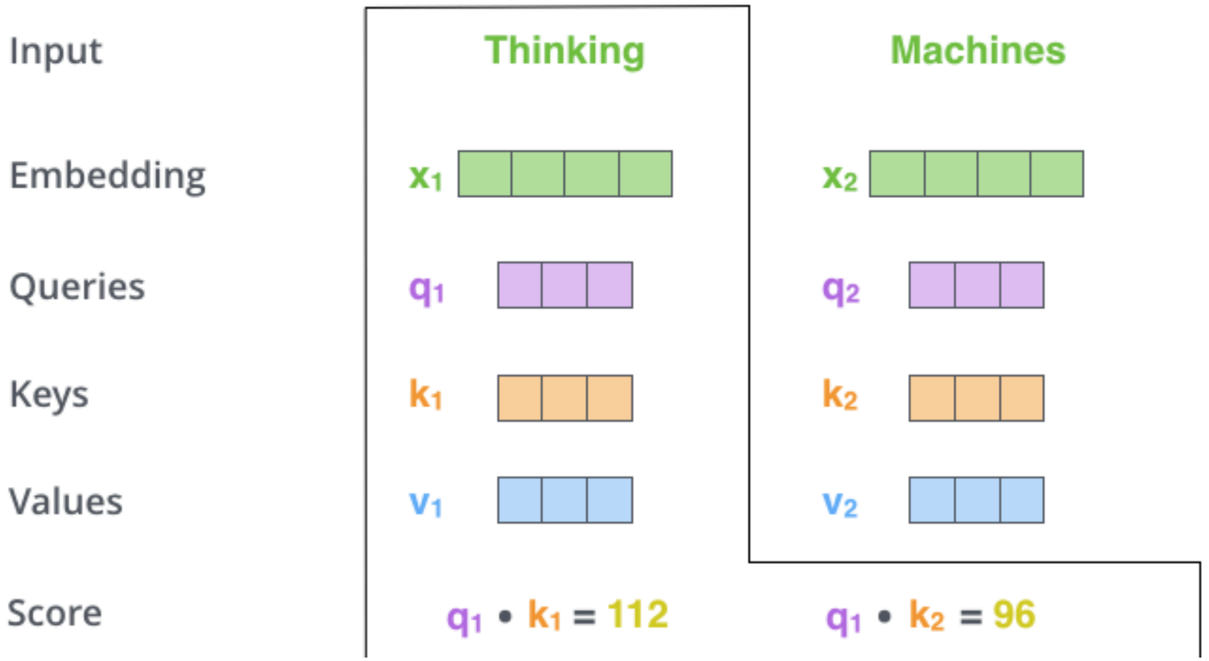
- Input latent vectors: $x_1, \ldots, x_n$
- Self-attention parameters: W^Q, W^K, W^V (weights for query, key, value)
- For each word i, compute
  - Query vector: $q_i = x_i W^Q$
  - Key vector: $k_i = x_i W^K$
  - Value vector: $v_i = x_i W^V$



Transformer

Self-attention layer

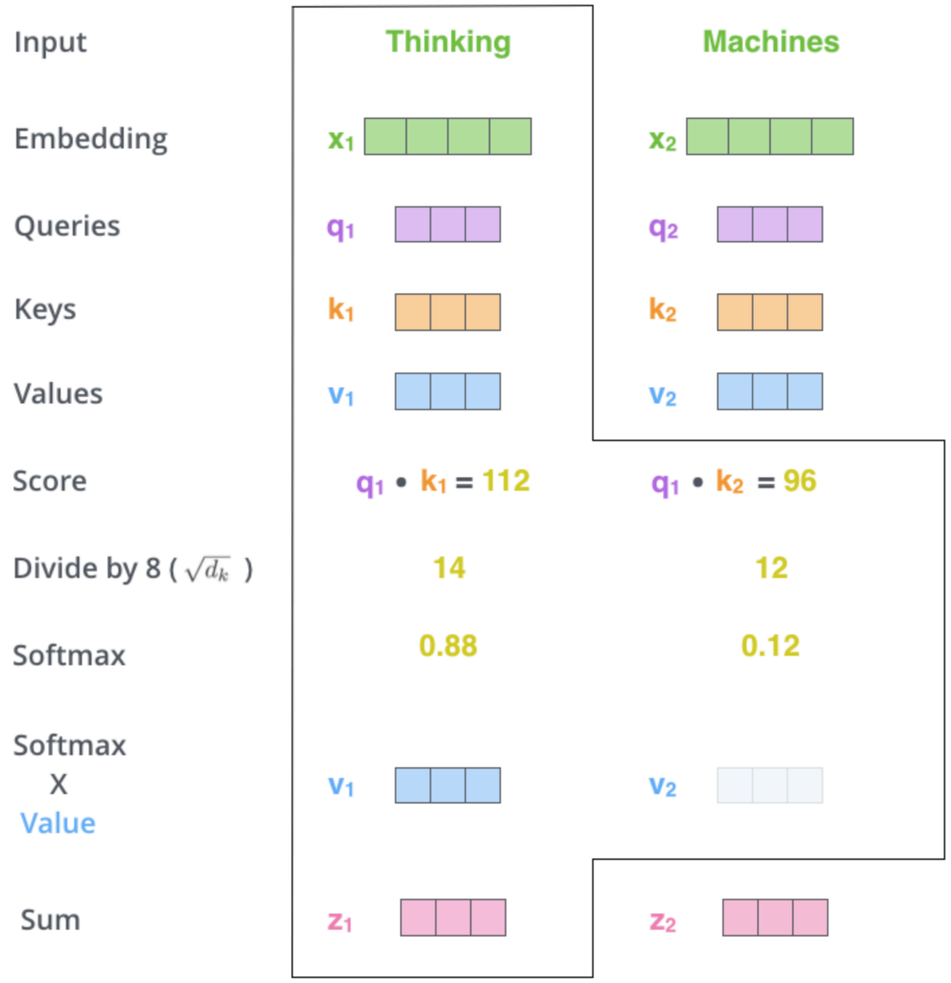
- For each word i, compute the scores to determine how much focus to place on other input words
  - The attention score for word j to word i: $q_i^T k_j$



Transformer

Self-attention layer

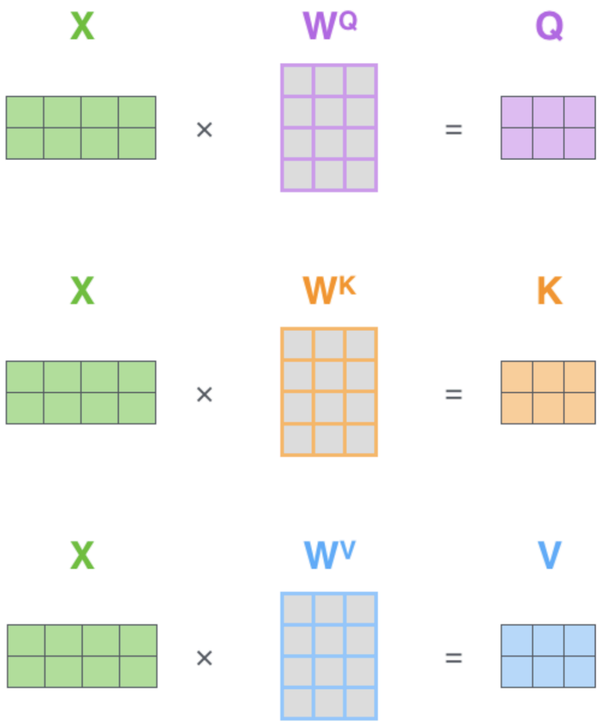
- For each word i, the output vector
  - $\sum_j s_{ij} v_j, \quad s_i = \text{softmax}(q_i^T k_1, \ldots, q_i^T k_n)$

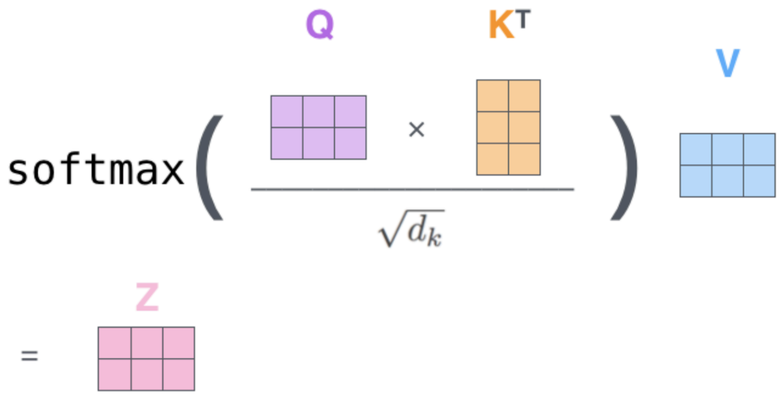


Transformer

Matrix form

- $Q = XW^Q,\ K = XW^K,\ V = XW^V,\ \ Z = \text{softmax}(QK^T)V$

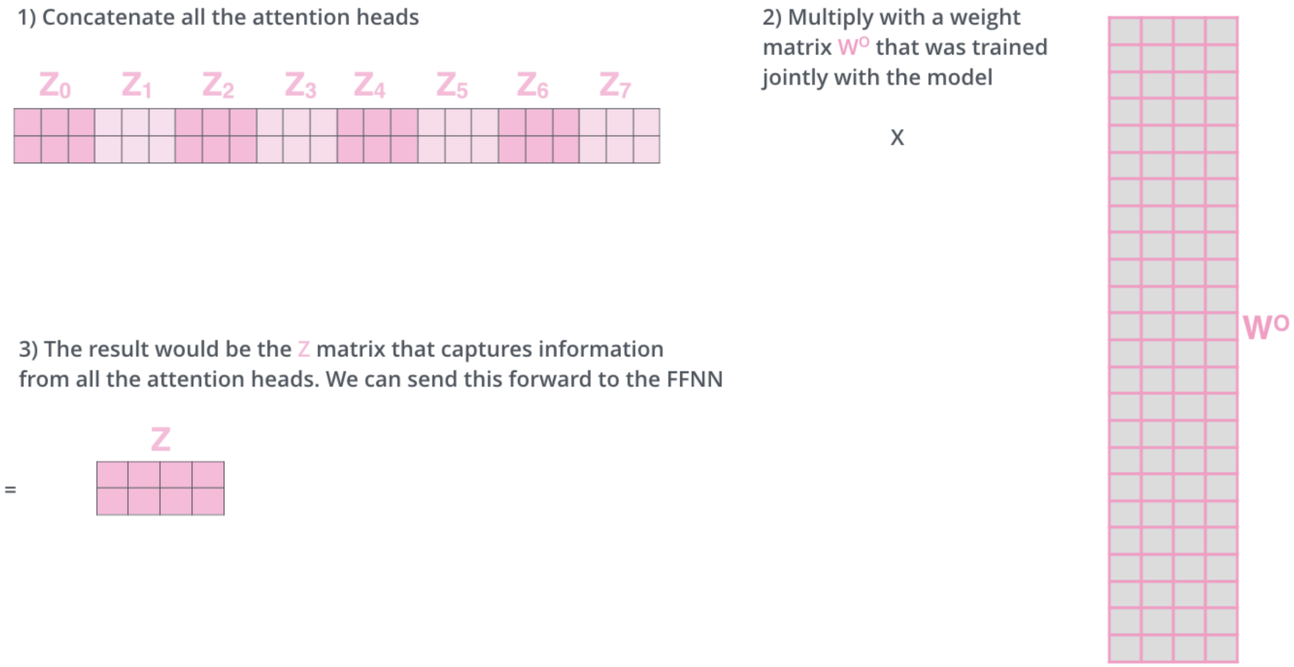




Transformer

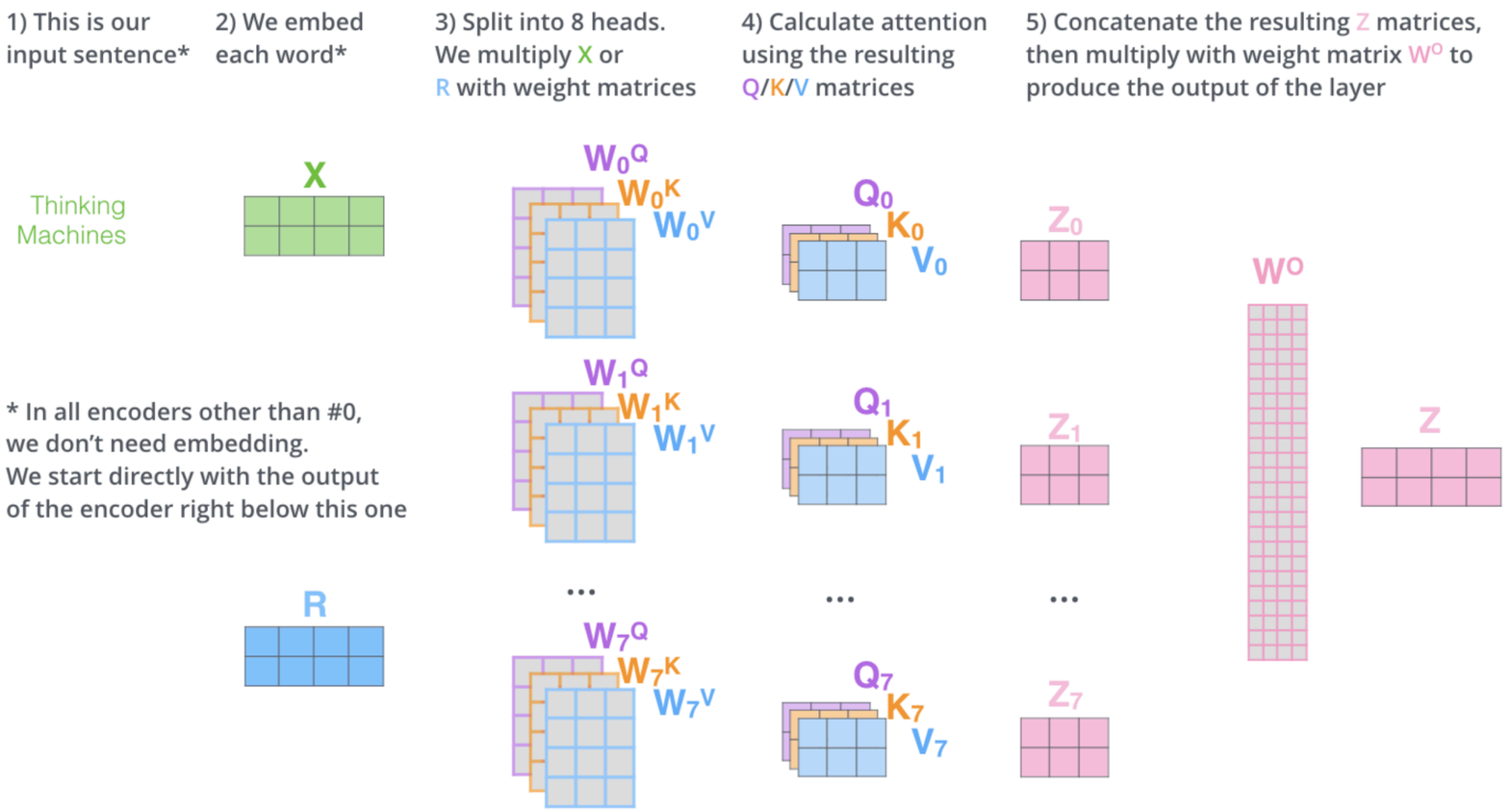
Multiply with weight matrix to reshape

- Gather all the outputs $Z_1, \ldots, Z_k$
- Multiply with a weight matrix to reshape
- Then pass to the next fully connected layer



Transformer

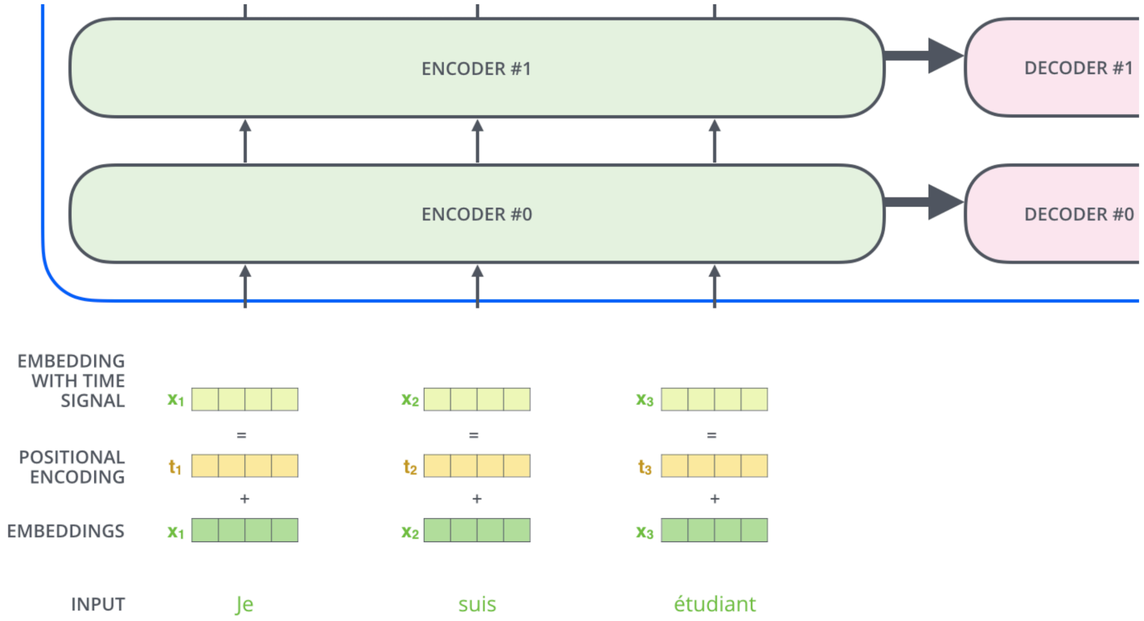
Overall architecture



Transformer

Sinusoidal Position Encoding

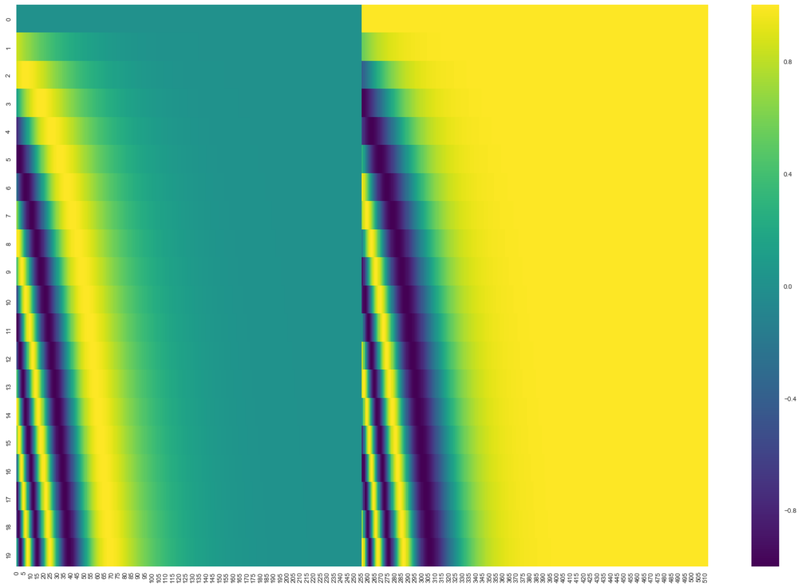
- The above architecture ignores the sequential information
- Add a positional encoding vector to each x_i (according to i)



Transformer

Positional Embedding

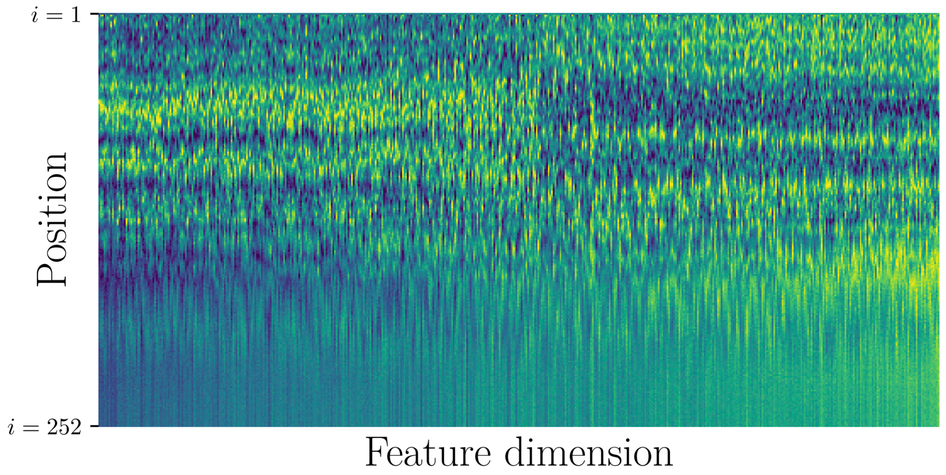
- Sin/cosine functions with different wavelengths (used in the original Transformer)
- The jth dimension of ith token $p_i[j] = \begin{cases} \sin(i \cdot c^{\frac{j}{d}}) \text{ if j is even} \\ \cos(i \cdot c^{\frac{j-1}{d}}) \text{ if j is odd} \end{cases}$
- smooth, parameter-free, inductive



Transformer

Types of positional encoding

- Position embedding: learn a latent vector for each position
  - non-smooth, data-driven (learnable), non-inductive
- Relative position embedding:
  - For each i, j, use the relative position embedding a_{j-i}
  - non-smooth, data-driven (learnable), (partial)-inductive

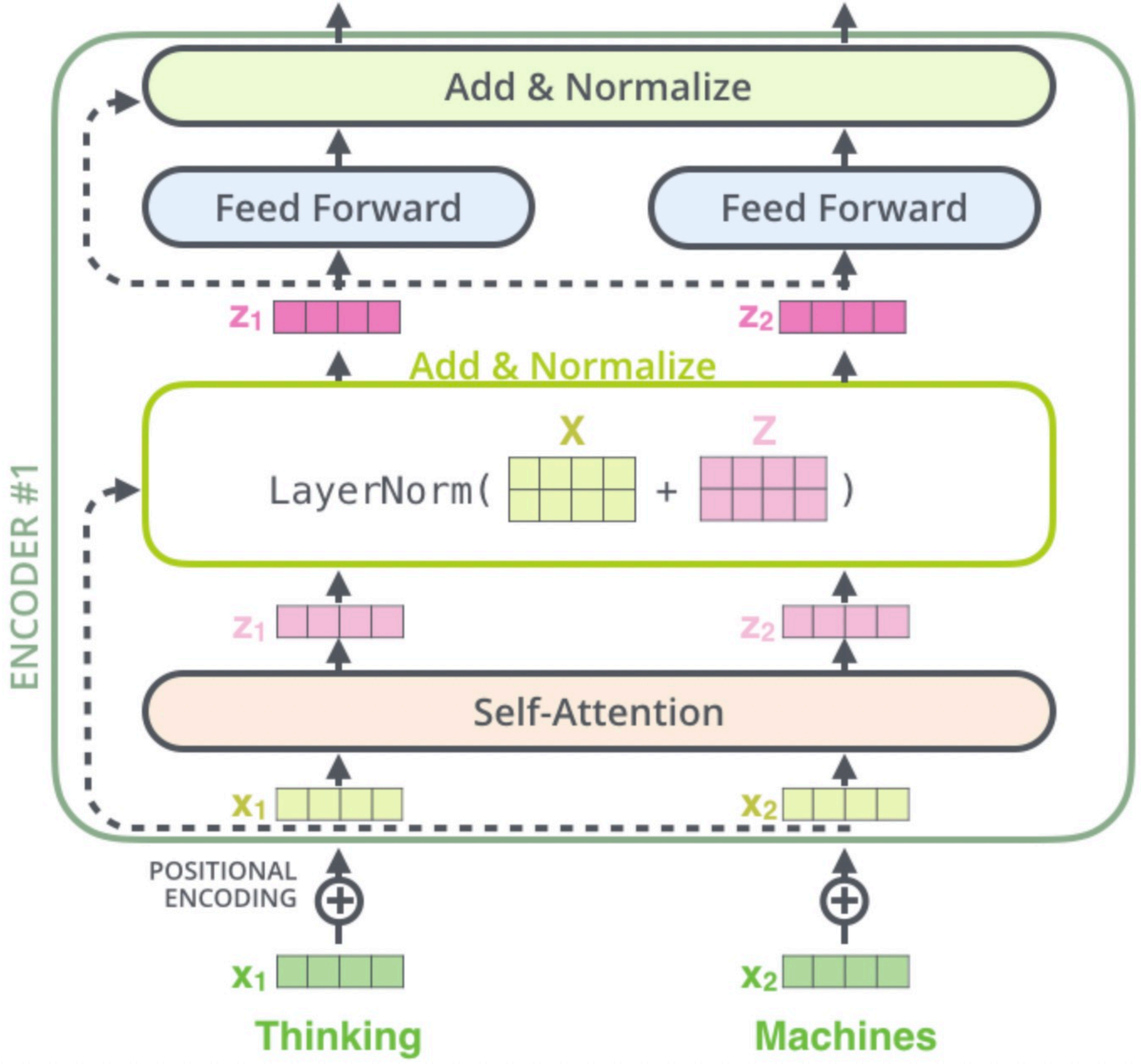


Transformer
Positional Encoding

- Neural ODE embedding :
  - Model positional embedding as a dynamic linear system
    - ``Learning to Encode Position for Transformer with Continuous Dynamical Model. Liu et al., 2020''
- Learnable Fourier Feature (Li et al., 2021):
  - $p_x = \phi(r_x, \theta)W_p$, where $r_x = \frac{1}{D}[\cos xW_r, \sin xW_r]$
  - W_r, W_p: learnable parameters (irrelevant to sequence length)
  - ``Learnable Fourier Features for Multi-Dimensional Spatial Positional Encoding. Li et al., 2021''
  - smooth, data-driven (learnable), inductive

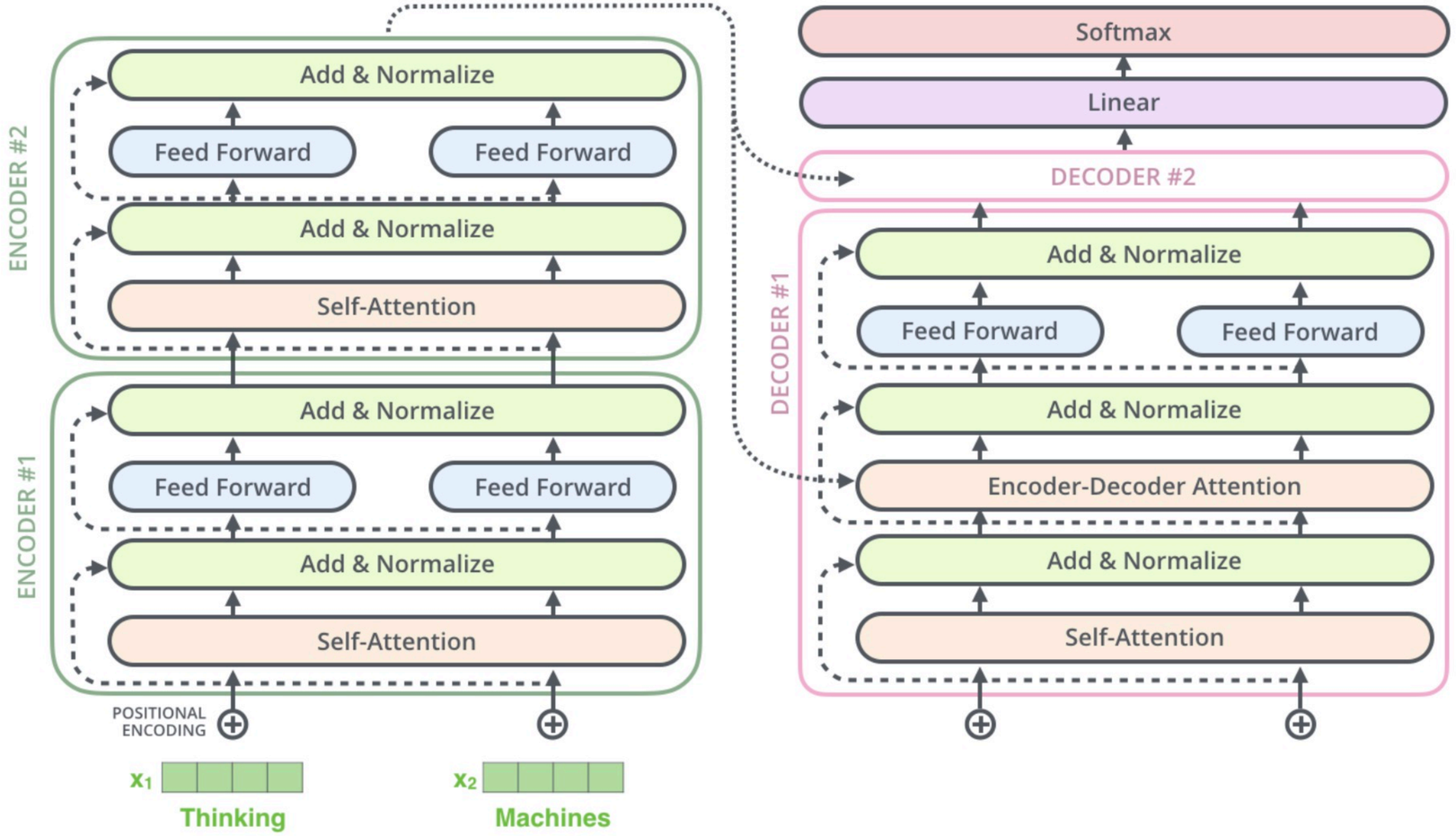
Transformer

Residual



Transformer

Whole framework



Unsupervised pertaining for NLP

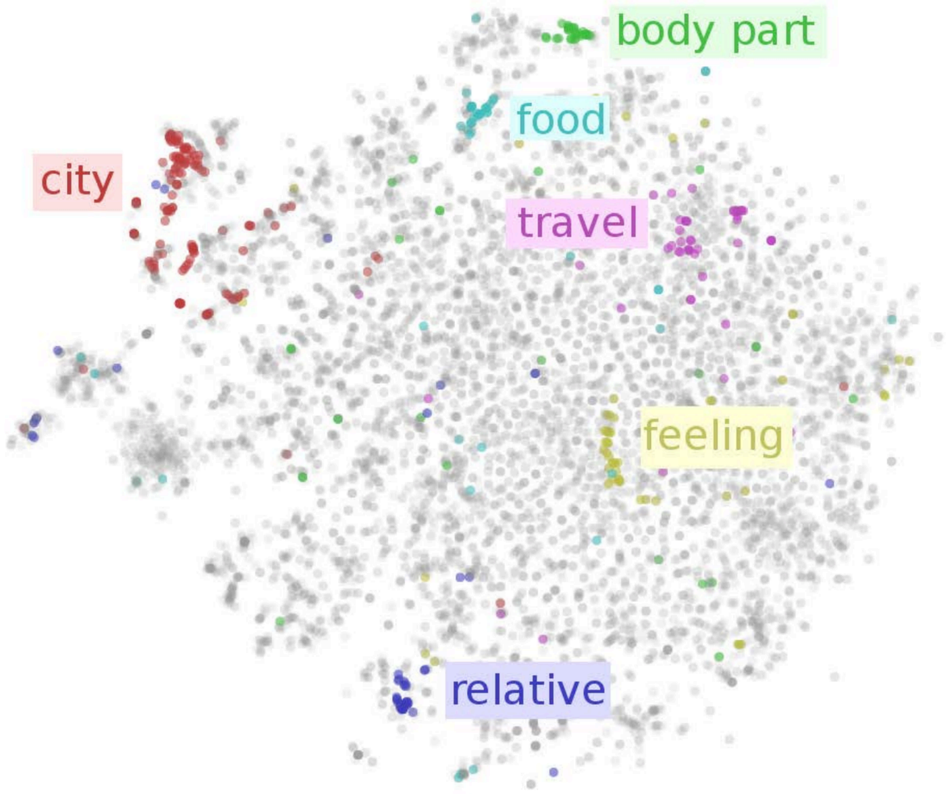
Motivation

- Many unlabeled NLP data but very few labeled data
- Can we use large amount of unlabeled data to obtain meaningful representations of words/sentences?

Unsupervised pertaining for NLP

Learning word embeddings

- Use large (unlabeled) corpus to learn a useful word representation
  - Learn a vector for each word based on the corpus
  - Hopefully the vector represents some semantic meaning
  - Can be used for many tasks
    - Replace the word embedding matrix for DNN models for classification/translation
  - Two different perspectives but led to similar results:
    - Glove (Pennington et al., 2014)
    - Word2vec (Mikolov et al., 2013)



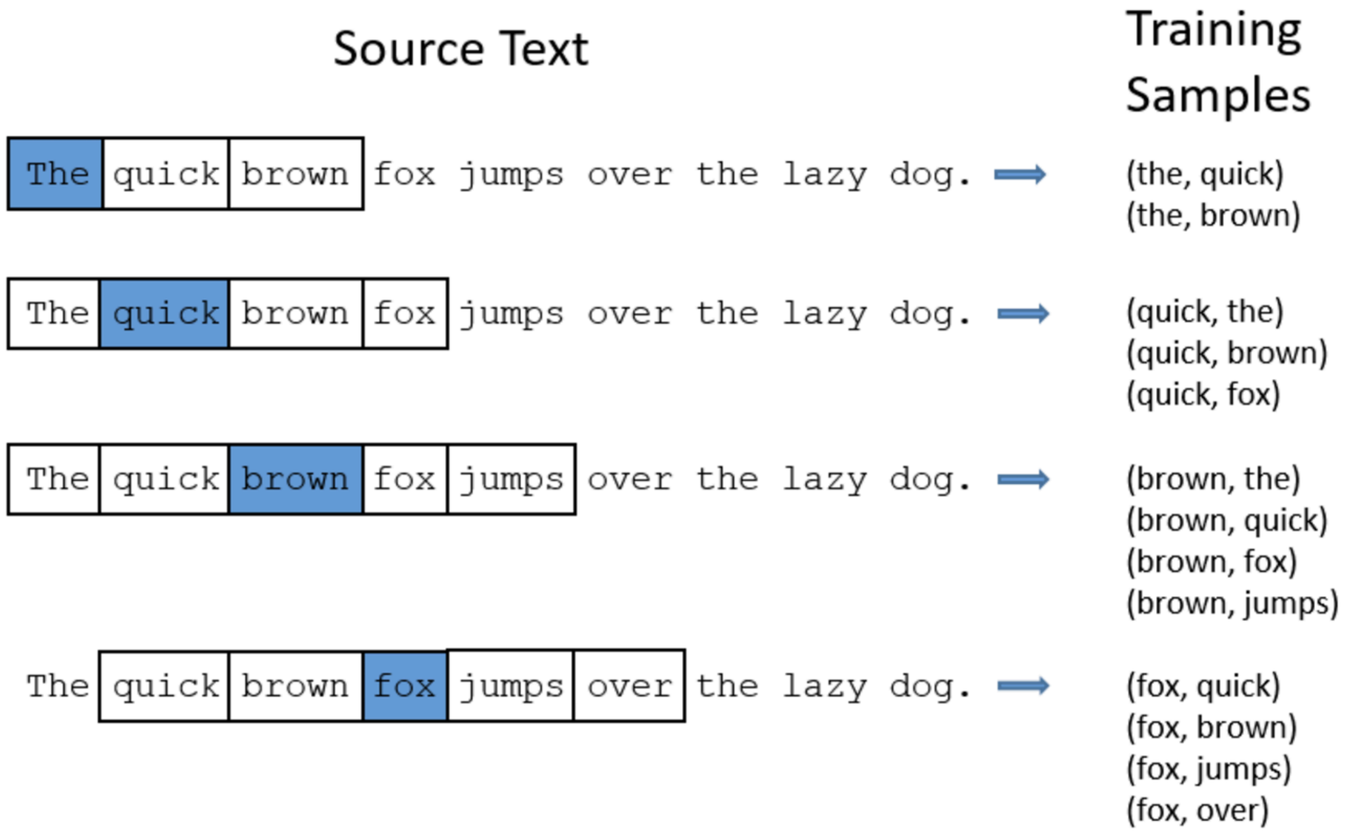
Unsupervised pertaining for NLP

Context information

- Given a large text corpus, how to learn low-dimensional features to represent a word?
- For each word w_i, define the ``contexts'' of the word as the words surrounding it in an L-sized window:
- $$w_{i-L-2}, w_{i-L-1}, \underbrace{w_{i-L}, \cdots, w_{i-1}}_{\text{contexts of } w_i}, w_i, \underbrace{w_{i+1}, \cdots, w_{i+L}}_{\text{contexts of } w_i}, w_{i+L+1}, \cdots$$
- Get a collection of (word, context) pairs, denoted by D.

Unsupervised pertaining for NLP

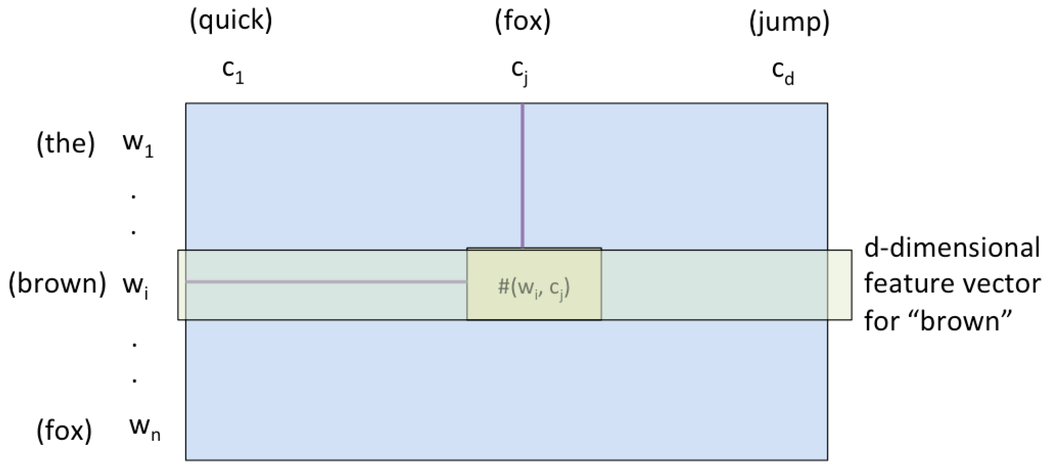
Examples



Unsupervised pertaining for NLP

Use bag-of-word model

- Idea 1: Use the bag-of-word model to ``describe'' each word
- Assume we have context words $c_1, \cdots, c_d$ in the corpus, compute
  - $\#(w, c_i) :=$ number of times the pair(w, c_i) appears in D
- For each word w, form a d -dimensional (sparse) vector to describe w
  - $\#(w, c_1), \cdots, \#(w, c_d),$



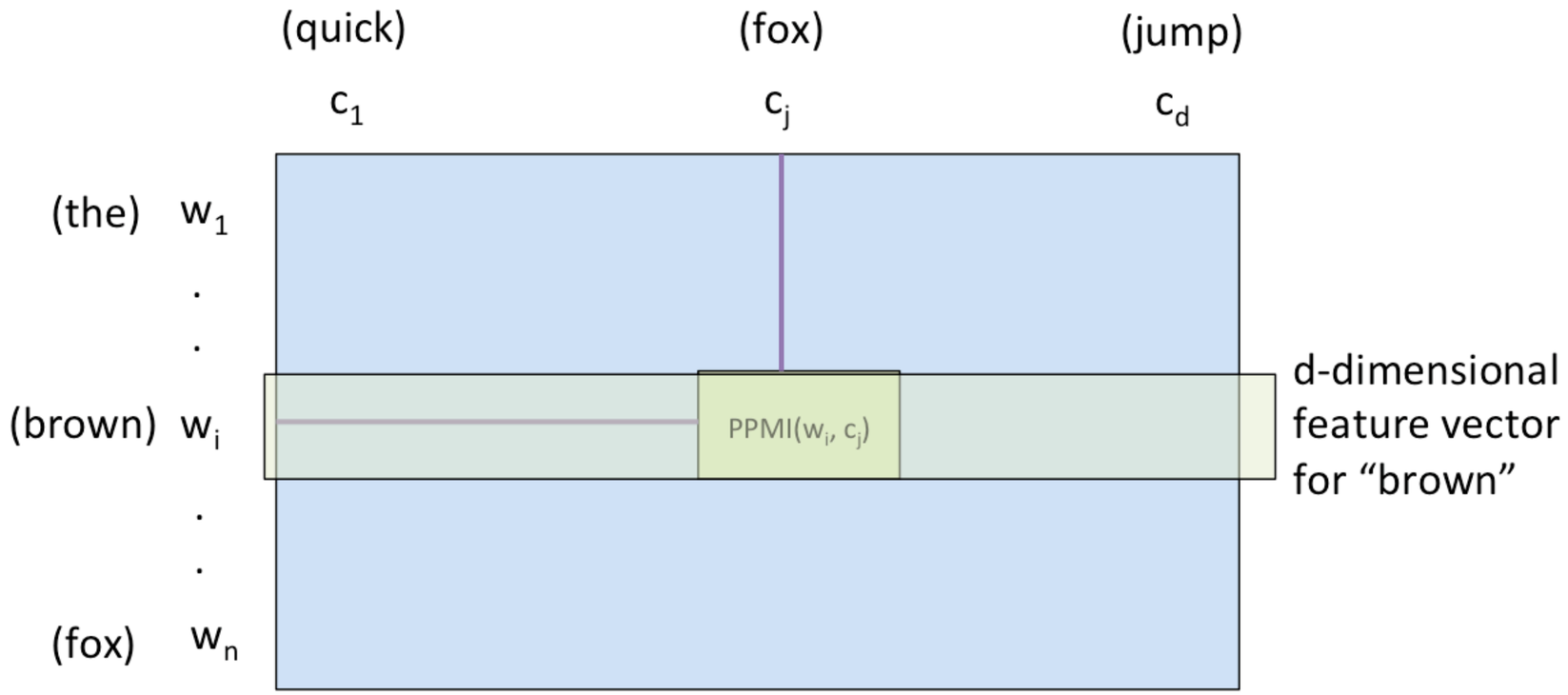
Unsupervised pertaining for NLP

PMI/PPMI Representation

- Similar to TF-IDF: Need to consider the frequency of each word and each context
- Instead of using co-ocurrent count $\#(w, c)$, we can define pointwise mutual information:
  - $\text{PMI}(w, c) = \log(\frac{\hat{P}(w, c)}{\hat{P}(w)\hat{P}(c)}) = \log \frac{\#(w, c)|D|}{\#(w)\#(c)}$,
  - $\#(w) = \sum_c \#(w, c)$: number of times word w occurred in D
  - $\#(c) = \sum_w \#(w, c)$: number of times context c occurred
  - $|D|$: number of pairs in D
- Positive PMI (PPMI) usually achieves better performance:
  - $\text{PPMI}(w, c) = \max(\text{PMI}(w, c), 0)$
- M^{PPMI}: a n by d word feature matrix, each row is a word and each column is a context

Unsupervised pertaining for NLP

PPMI Matrix



Unsupervised pertaining for NLP

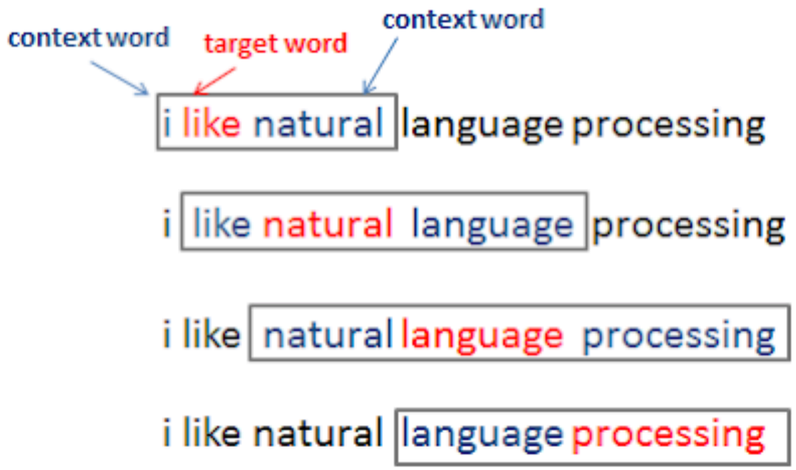
Generalized Low-rank Embedding

- SVD basis will minimize
  - $\min_{W,V} \|M^{\text{PPMI}} - WV^T\|_F^2$
- Glove (Pennington et al., 2014)
  - Negative sampling (less weights to 0s in M^{PPMI})
  - Adding bias term:
    - $M^{\text{PPMI}} \approx WV^T + b_w e^T + e b_c^T$
- Use W or V as the word embedding matrix

Unsupervised pertaining for NLP

Word2vec (Mikolov et al., 2013)

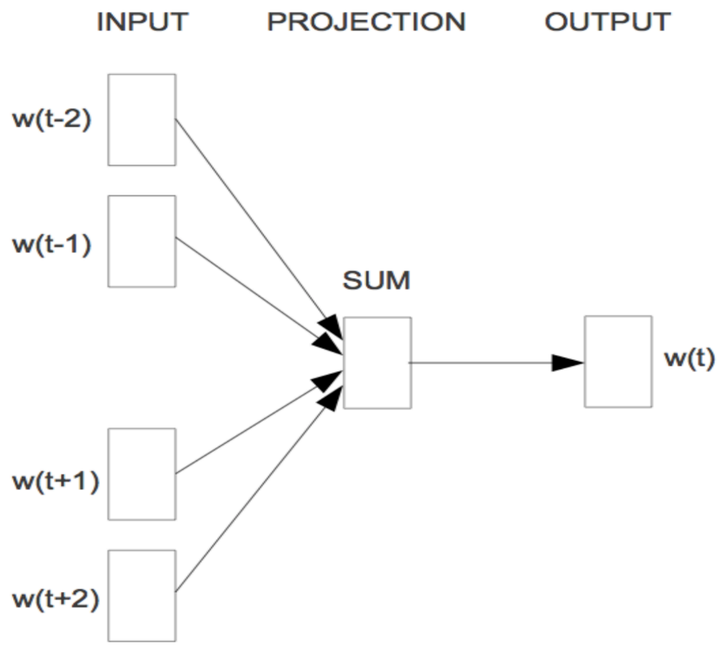
- A neural network model for learning word embeddings
- Main idea:
  - Predict the target words based on the neighbors (CBOW)
  - Predict neighbors given the target words (Skip-gram)



Unsupervised pertaining for NLP

CBOW (Continuous Bag-of-Word model)

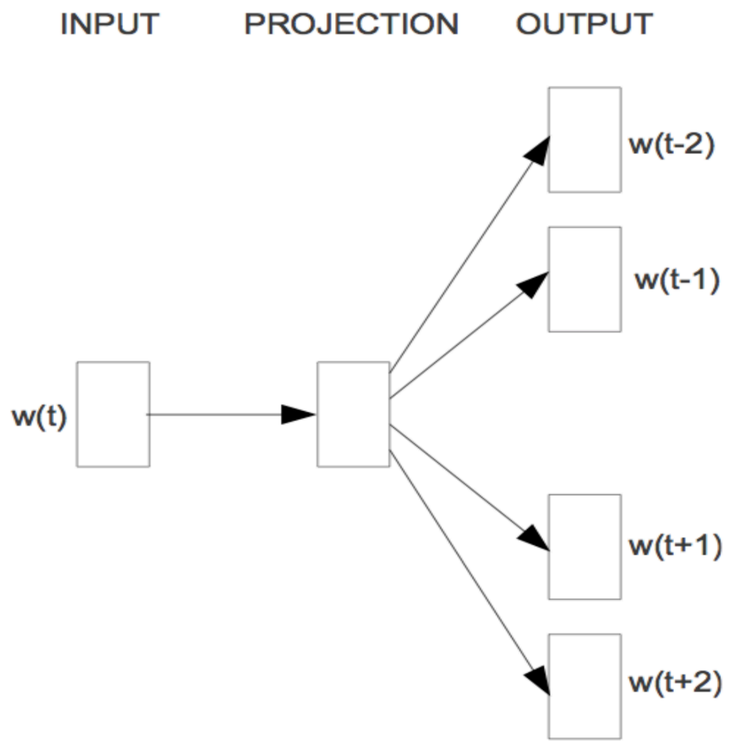
- Predict the target words based on the neighbors



Unsupervised pertaining for NLP

Skip-gram

- Predict neighbors using target word



Unsupervised pertaining for NLP

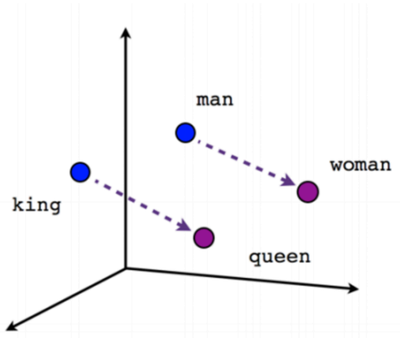
More on skip-gram

- Learn the probability $P(w_{t+j} \mid w_t)$: the probability to see w_{t+j} in target word w_t's neighborhood
- Every word has two embeddings:
  - v_i serves as the role of target
  - u_i serves as the role of context
- Model probability as softmax:
  - $$P(o \mid c) = \frac{e^{u_o^T v_c}}{\sum_{w=1}^{W} e^{u_w^T v_c}}$$

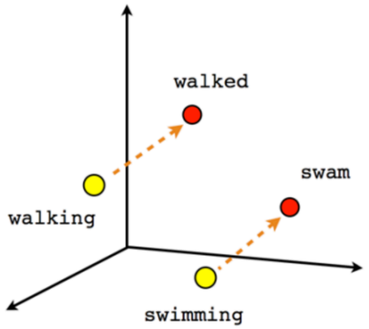
Unsupervised pertaining for NLP

Results

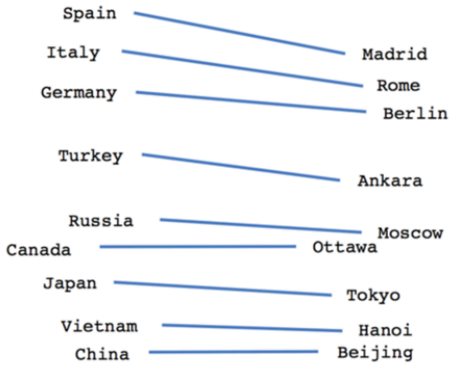
- The low-dimensional embeddings are (often) meaningful:



Male-Female



Verb tense



Country-Capital

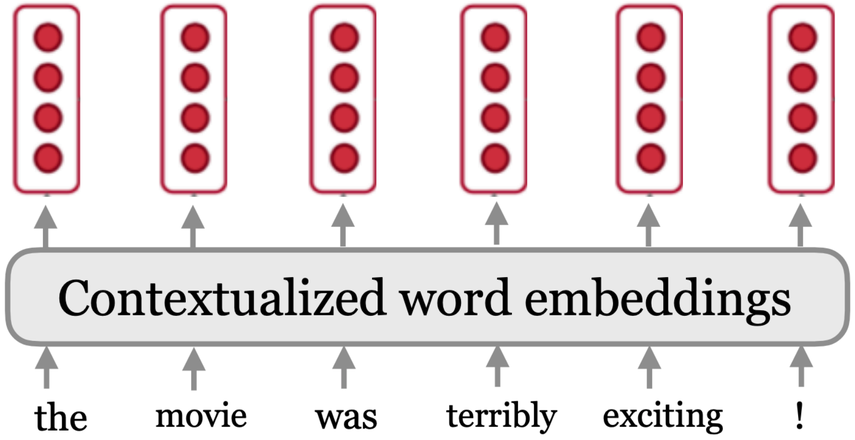
Contextual embedding

Contextual world representation

- The semantic meaning of a word should depend on its context

```
[0.9, -0.2, 1.6, …]                    [-1.9, -0.4, 0.1, …]
        ↑                                      ↑
open a bank account                  on the river bank
```

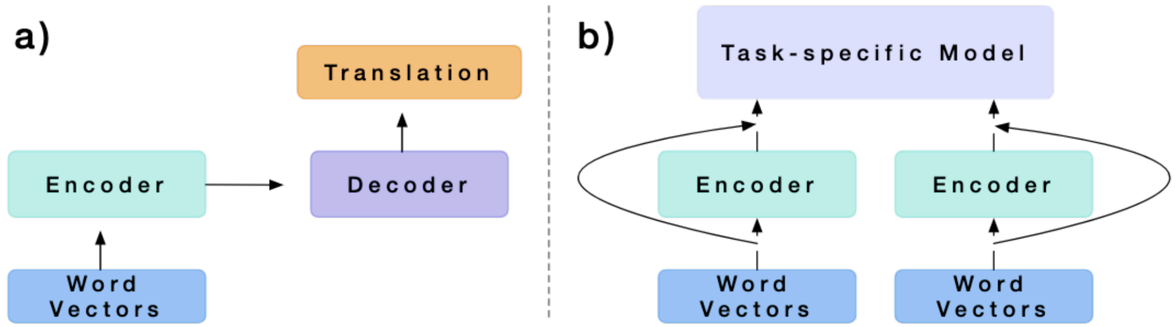
- Solution: Train a model to extract contextual representations on text corpus



Contextual embedding

CoVe (McCann et al., 2017)

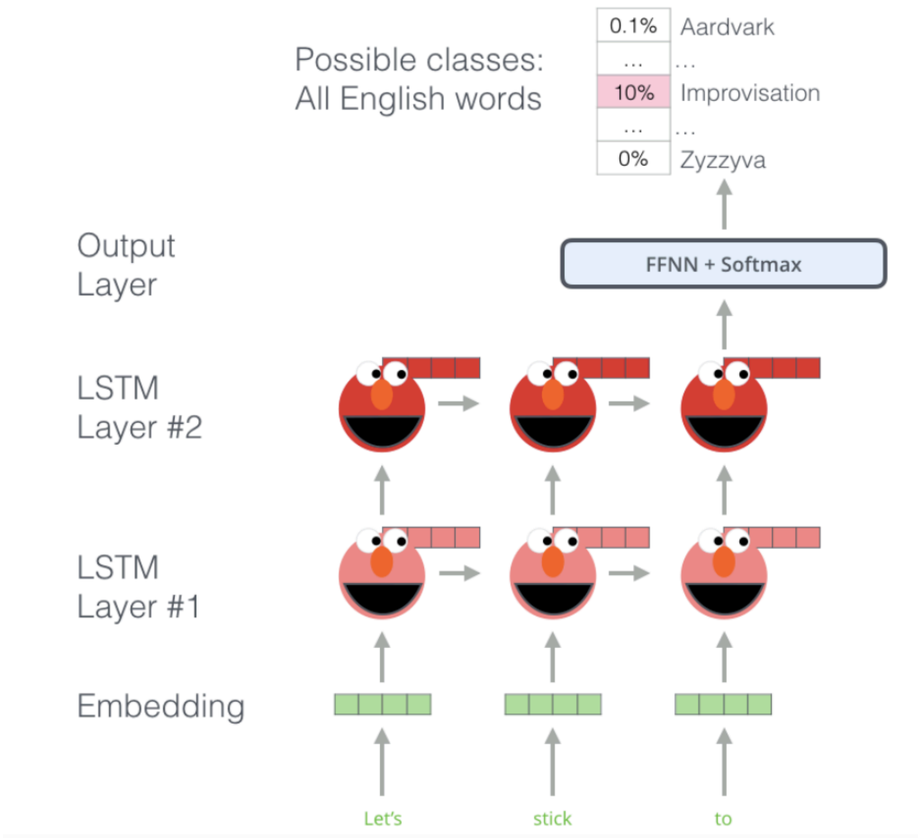
- Key idea: Train a standard neural machine translation model
- Take the encoder directly as contextualized word embeddings
- Problems:
  - Translation requires paired (labeled) data
  - The embeddings are tailored to particular translation corpuses



Contextual embedding

Language model pretraining task

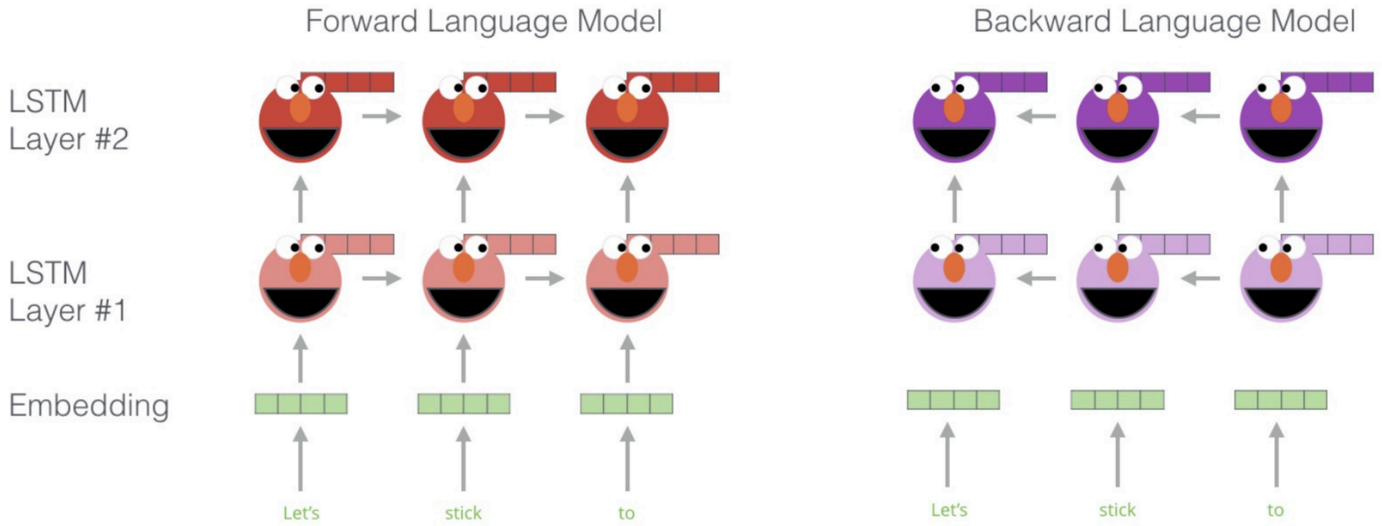
- Predict the next word given the prefix
- Can be defined on any unlabeled document



Contextual embedding

ELMo (Peter et al., 2018)

- Key ideas:
  - Train a foward and backward LSTM language model on large corpus
  - Use the hidden states for each token to compute a vector representation of each word
  - Replace the word embedding by Elmo's embedding (with fixed Elmo's LSTM weights)



Contextual embedding

ELMo results

Task	Previous SOTA		Our baseline	ELMo + baseline	Increase (absolute/ relative)
SQuAD	Liu et al. (2017)	84.4	81.1	85.8	4.7 / 24.9%
SNLI	Chen et al. (2017)	88.6	88.0	88.7 ± 0.17	0.7 / 5.8%
SRL	He et al. (2017)	81.7	81.4	84.6	3.2 / 17.2%
Coref	Lee et al. (2017)	67.2	67.2	70.4	3.2 / 9.8%
NER	Peters et al. (2017)	91.93 ± 0.19	90.15	92.22 ± 0.10	2.06 / 21%
SST-5	McCann et al. (2017)	53.7	51.4	54.7 ± 0.5	3.3 / 6.8%

Contextual embedding

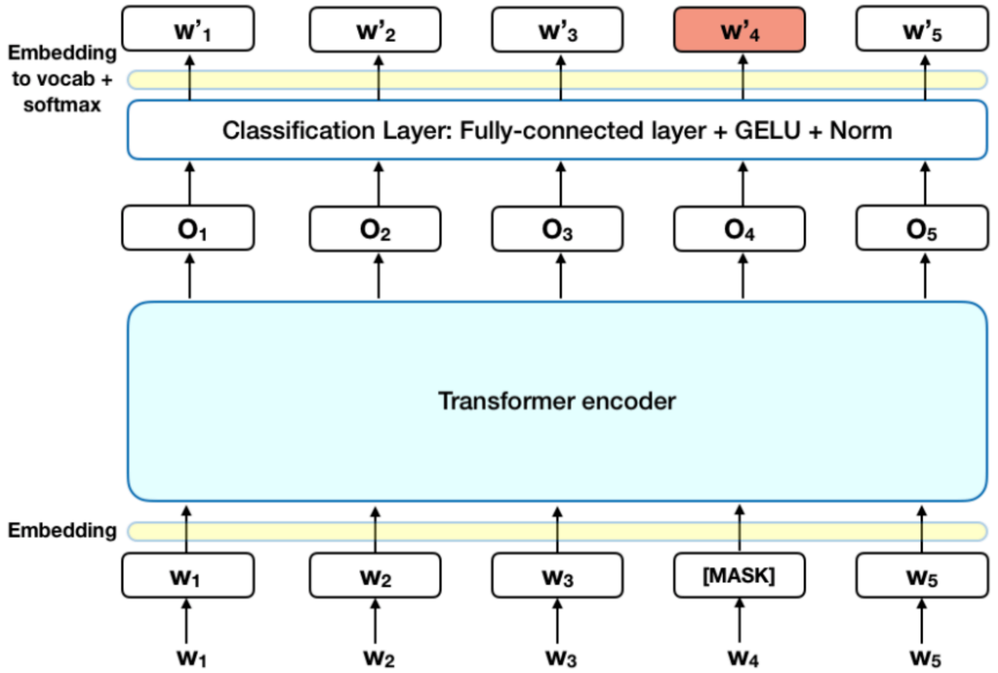
BERT

- Key idea: replace LSTM by Transformer
- Define the generated pretraining task by masked language model
- Two pretraining tasks
- Finetune both BERT weights and task-dependent model weights for each task

Contextual embedding

BERT pretraining loss

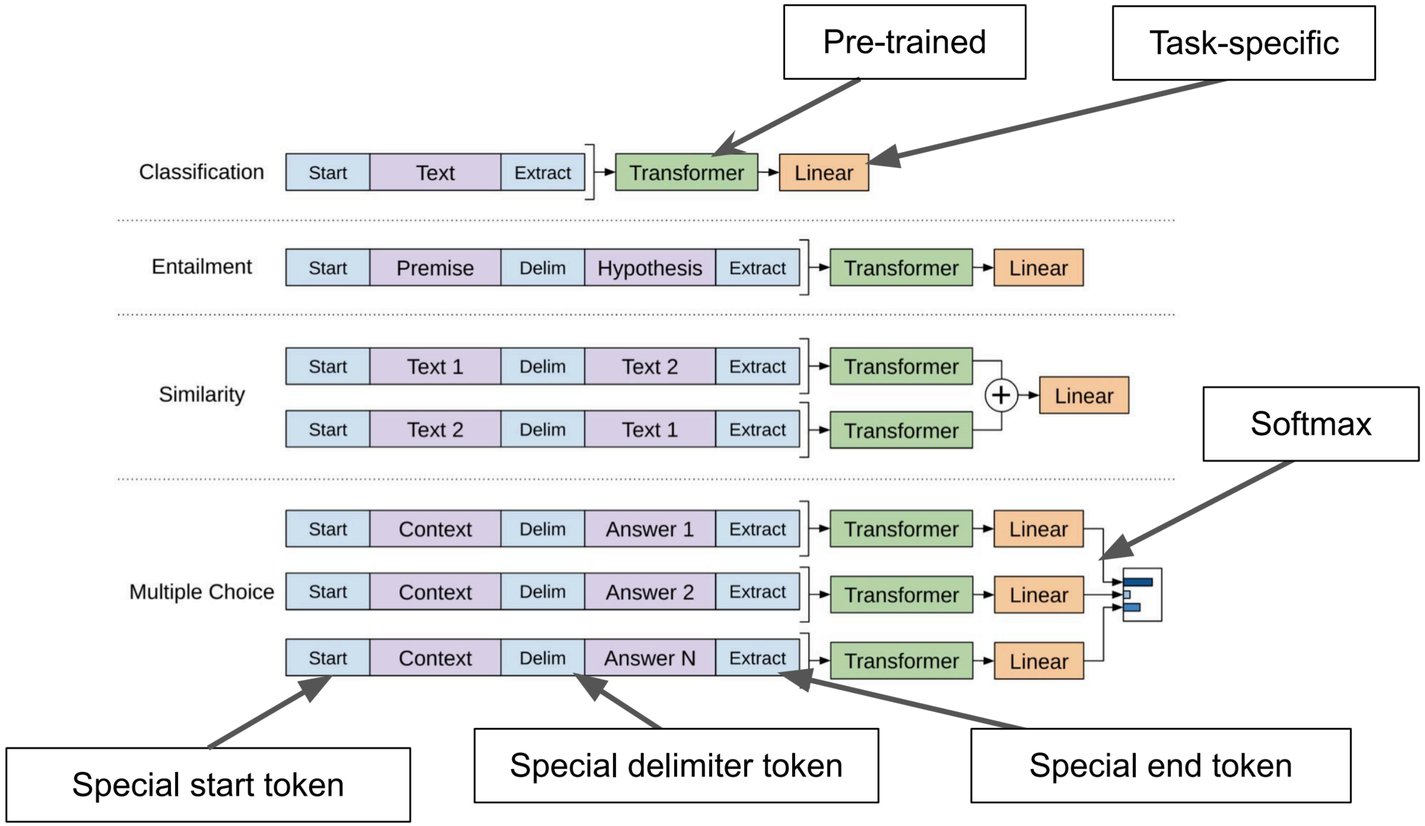
- Masked language model: predicting each word by the rest of sentence
- Next sentence prediction: the model receives pairs of sentences as input and learns to predict if the second sentence is the subsequent sentence in the original document.



Contextual embedding

BERT finetuning

- Keep the pretrained Transformers
- Replace or append a layer for the final task
- Train the whole model based on the task-dependent loss



Contextual embedding

BERT results

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**