COMP5212: Machine Learning

Lecture 15

Minhao Cheng

Representation for sentence/document

Bag of word + Fully connected network

- $f(x) = W_L\sigma(W_{L-1}\cdots\sigma(W_0x))$
- The first layer W_0 is a d_1 by d matrix:
  - Each column w_i is a d_1 dimensional representation of i-th word （word embedding)
  - $W_0x = x_1w_1 + x_2w_2 + \cdots + x_dw_d$ is a linear combination of these vectors
  - W_0 is also called the word embedding matrix
  - Final prediction can be viewed as an $L-1$ layer network on W_0x (average of word embeddings)
- Not capturing the sequential information

Recurrent Neural Network

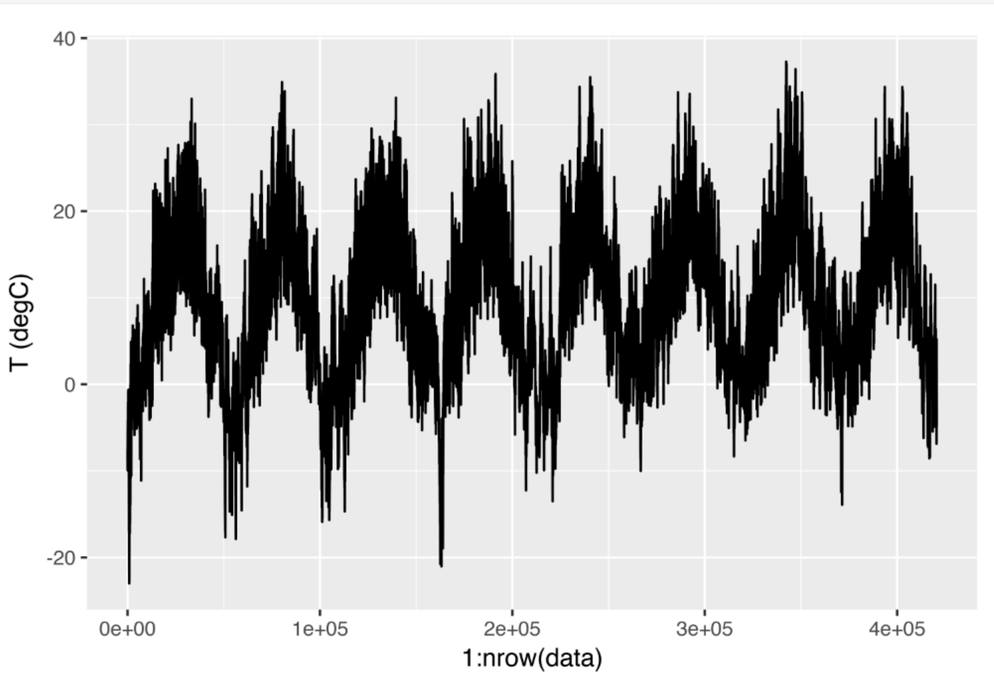
Time series/Sequence data

- Input: $\{x_1, x_2, \cdots, x_T\}$
  - Each x_t is the feature at time step t
  - Each x_t can be a d-dimensional vector
- Output: $\{y_1, y_2, \cdots, y_T\}$
  - Each y_t is the output at step t
  - Multi-class output or Regression output:
    - $y_t \in \{1,2,\cdots, L\}$ or $y_t \in \mathbb{R}$

Recurrent Neural Network

Example: Time Series Prediction

- Climate Data:
  - x_t: temperature at time t
  - y_t: temperature (or temperature change) at time $t + 1$
- Stock Price: Predicting stock price



Recurrent Neural Network

Example: Language Modeling

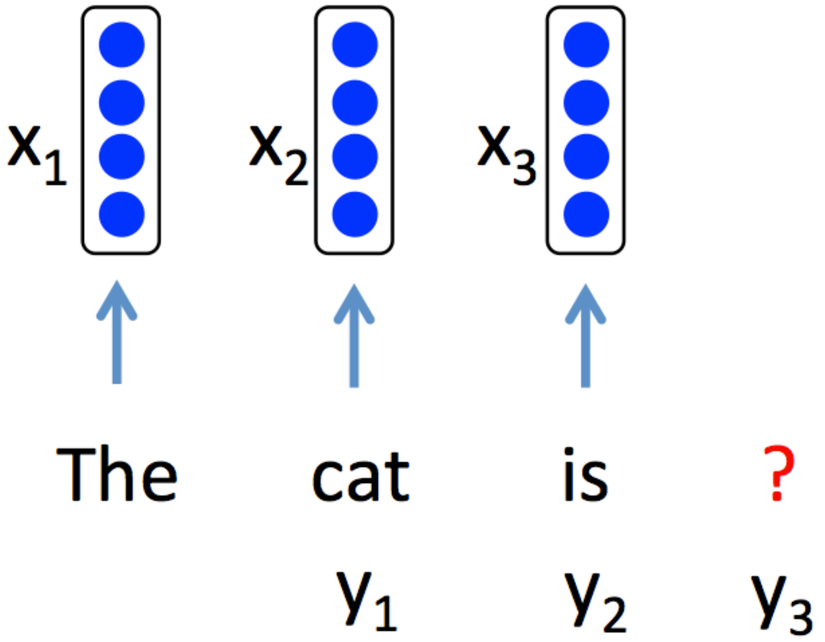
The cat is ?

Recurrent Neural Network

Example: Language Modeling

The cat is ?

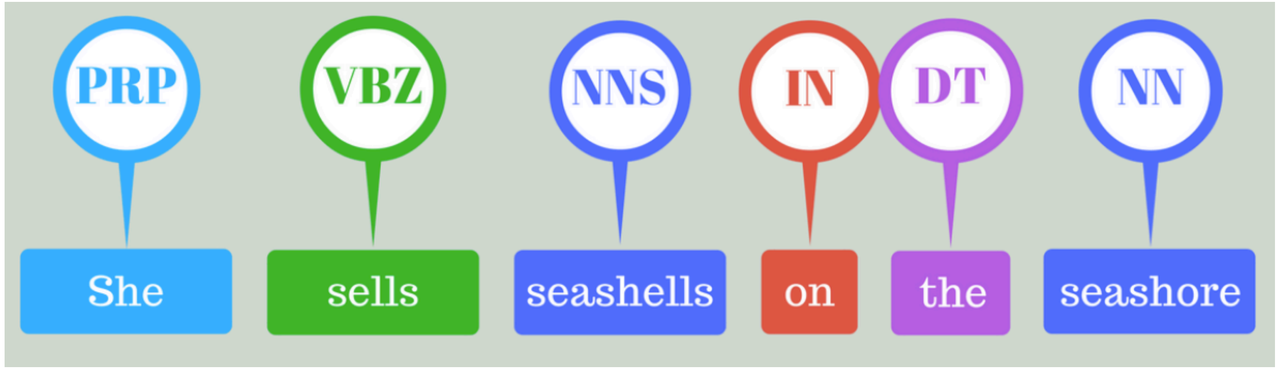
- x_t: one-hot encoding to represent the word at step t ($[0,\ldots,0,1,0,\ldots,0]$)
- y_t: the next word
  - $y_t \in \{1,\cdots,V\}$ V: Vocabulary size



Recurrent Neural Network

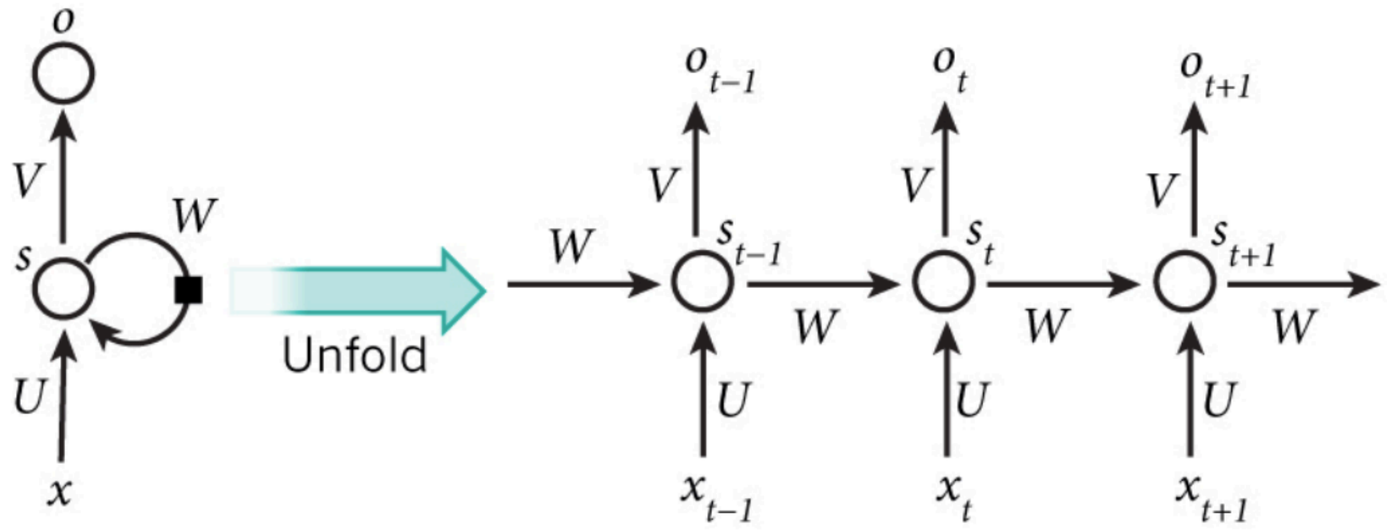
Example: POS Tagging

- Part of Speech Tagging:
  - Labeling words with their Part-Of-Speech (Noun, Verb, Adjective, …)
  - x_t: a vector to represent the word at step t
  - y_t: label of word t



Recurrent Neural Network

Example: POS Tagging



- x_t: t-th input
- s_t: hidden state at time t (`memory'' of the network)
  - $s_t = f(Ux_t + Ws_{t-1})$
  - W: transition matrix, U: word embedding matrix, s_0 usually set to be 0
- Predicted output at time t:
  - $o_t = \arg\max_i (Vs_t)_i$

Recurrent Neural Network

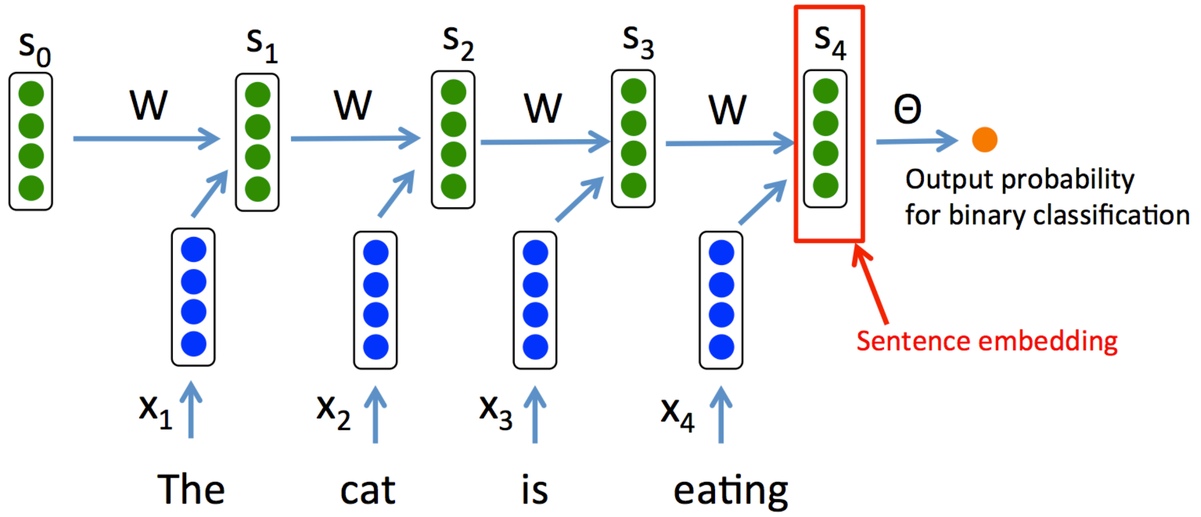
Recurrent Neural Network (RNN)

- Training: Find U, W, V to minimize empirical loss:
- Loss of a sequence:
  - $\sum_{t=1}^{T} \text{loss}(Vs_t, y_t)$
  - (s_t is a function of U, W, V)
- Loss on the whole dataset:
  - Average loss over all sequences
- Solved by SGD/Adam

Recurrent Neural Network

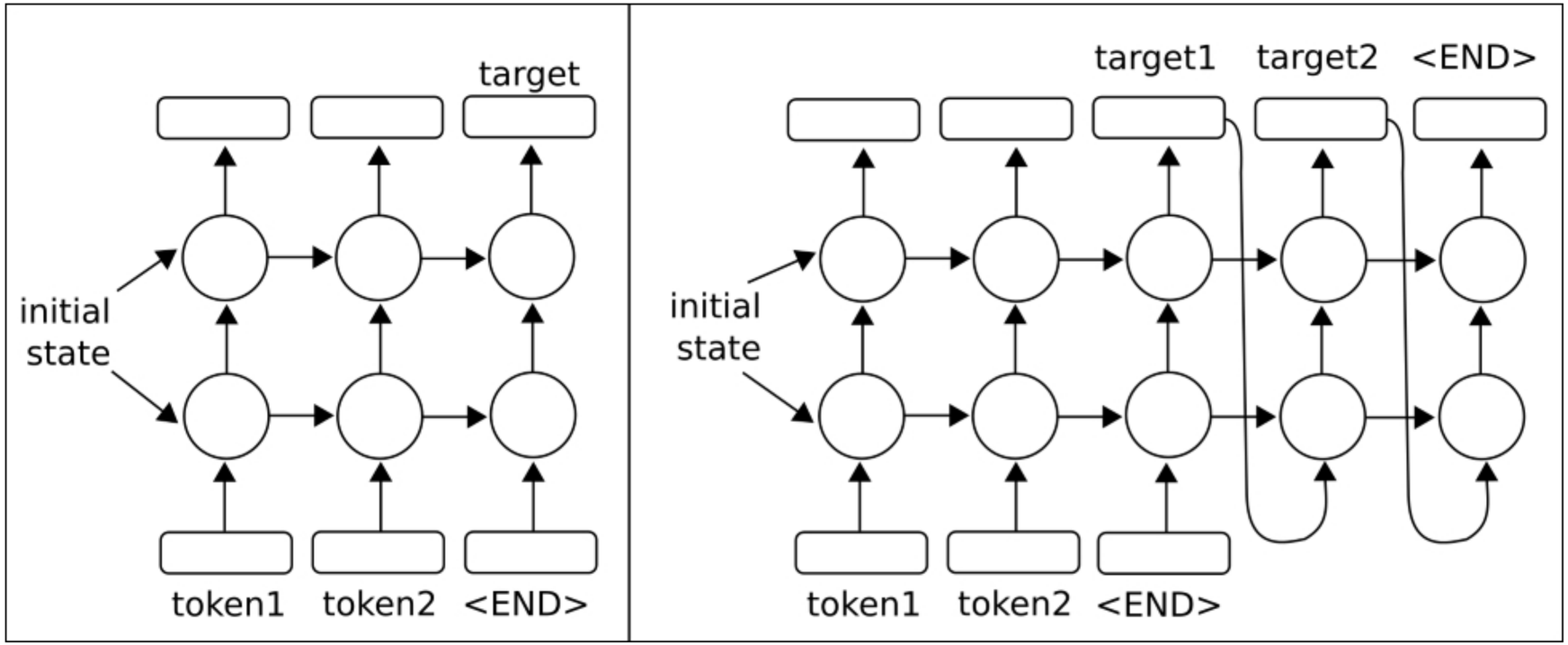
RNN: Text Classification

- Not necessary to output at each step
- Text Classification:
  - sentence → category
  - Output only at the final step
- Model: add a fully connected network to the final embedding



Recurrent Neural Network

Multi-layer RNN



Recurrent Neural Network

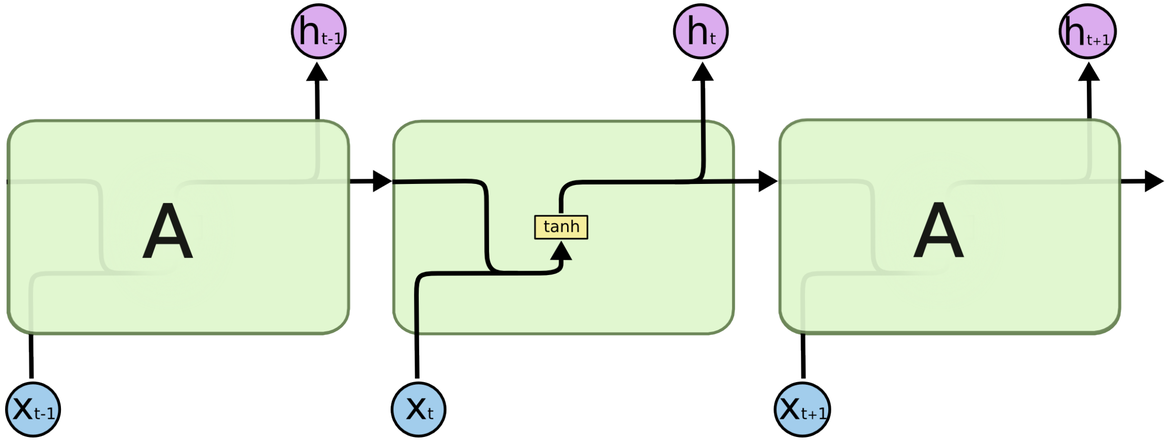
Problems of Classical RNN

- Hard to capture long-term dependencies
- Hard to solve (vanishing gradient problem)
- Solution:
  - LSTM (Long Short Term Memory networks)
  - GRU (Gated Recurrent Unit)
  - …

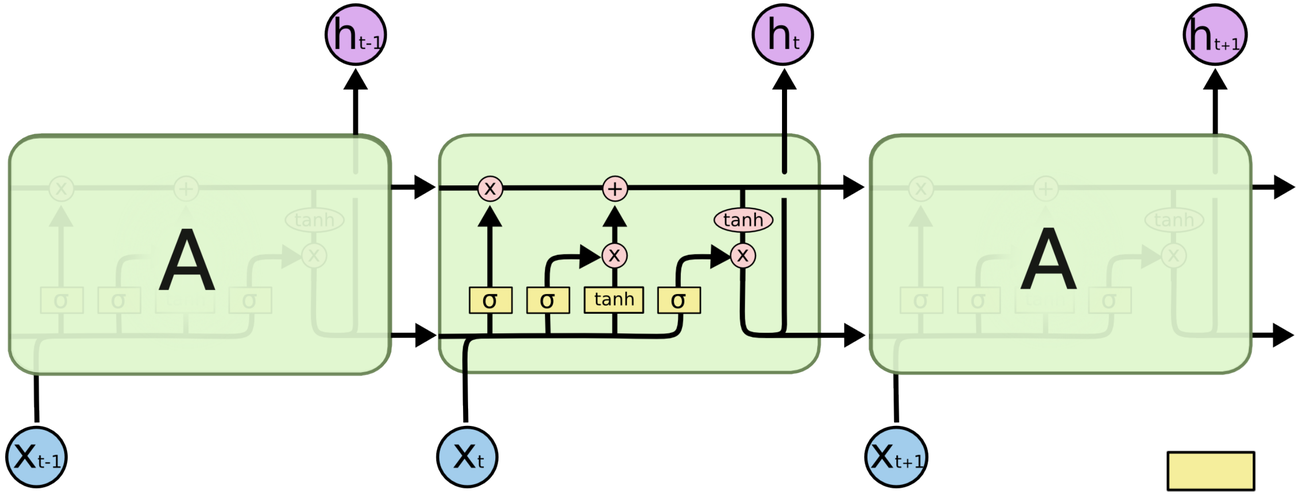
Recurrent Neural Network

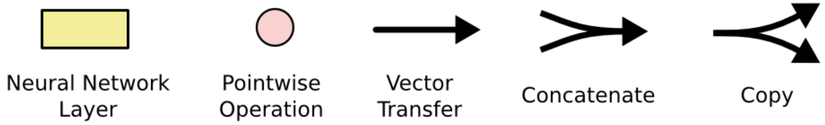
LSTM

- RNN:



- LSTM:

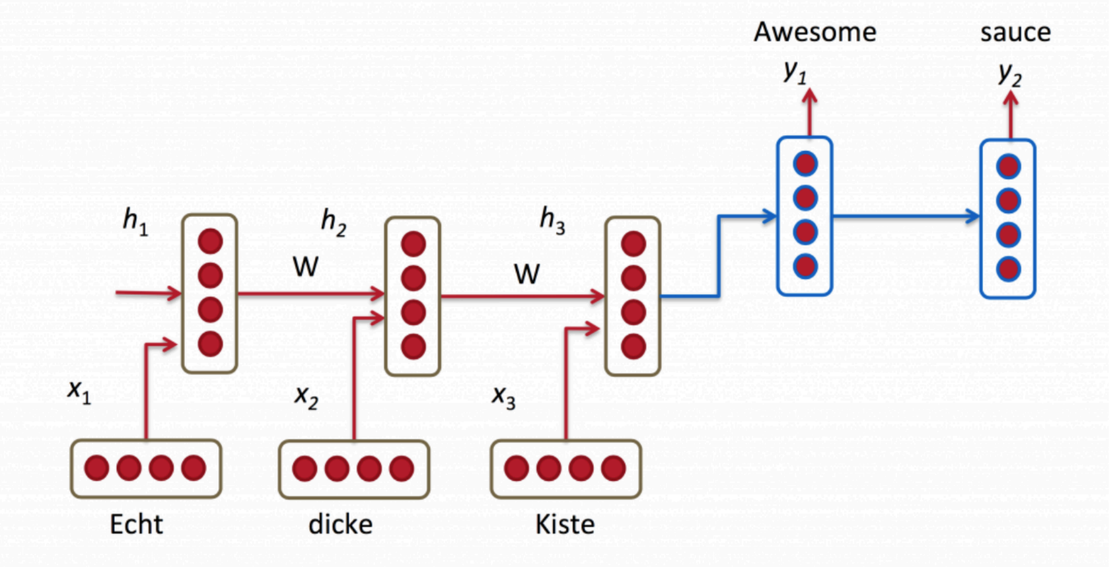




Recurrent Neural Network

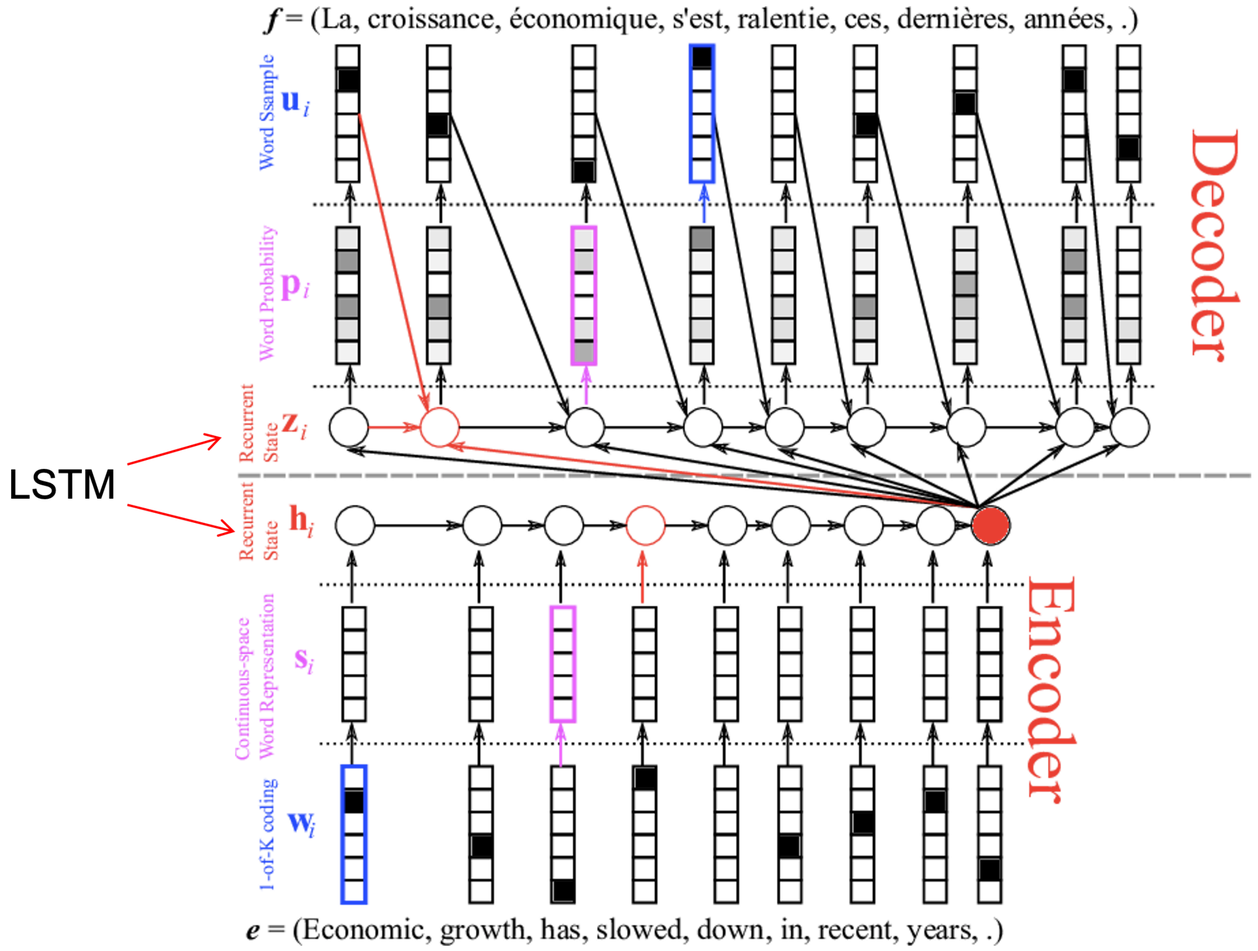
Neural Machine Translation (NMT)

- Out the translated sentence from an input sentence
- Training data: a set of input-output pairs (supervised setting)
- Encoder-decoder approach:
  - Encoder: Use (RNN/LSTM) to encode the input sentence input a latent vector
  - Decoder: Use (RNN/LSTM) to generate a sentence based on the latent vector



Recurrent Neural Network

Neural Machine Translation



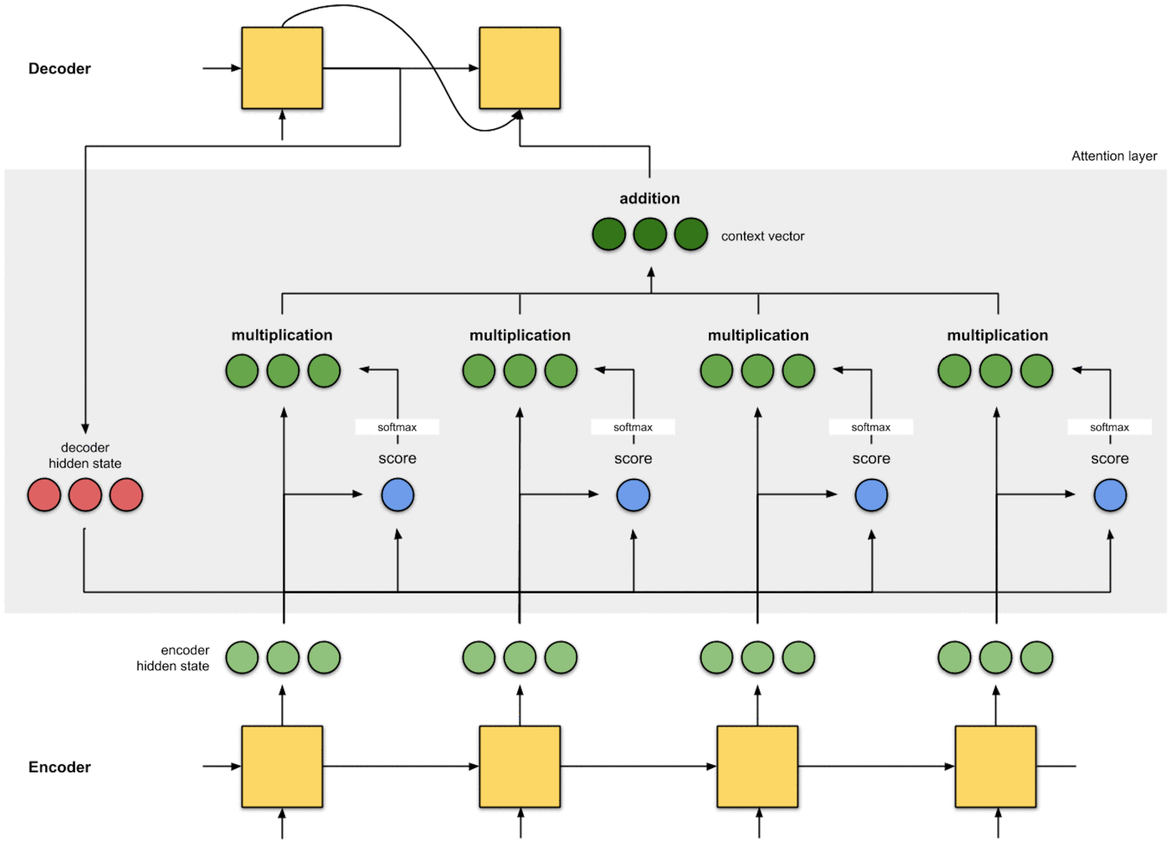
Recurrent Neural Network

Attention in NMT

- Usually, each output word is only related to a subset of input words (e.g., for machine translation)
- Let u be the current decoder latent state, $v_1, \ldots, v_n$ be the latent sate for each input word
- Compute the weight of each state by
  - $p = \text{Softmax}(u^T v_1, \ldots, u^T v_n)$
- Compute the context vector by $Vp = p_1 v_1 + \ldots + p_n v_n$

Transformer

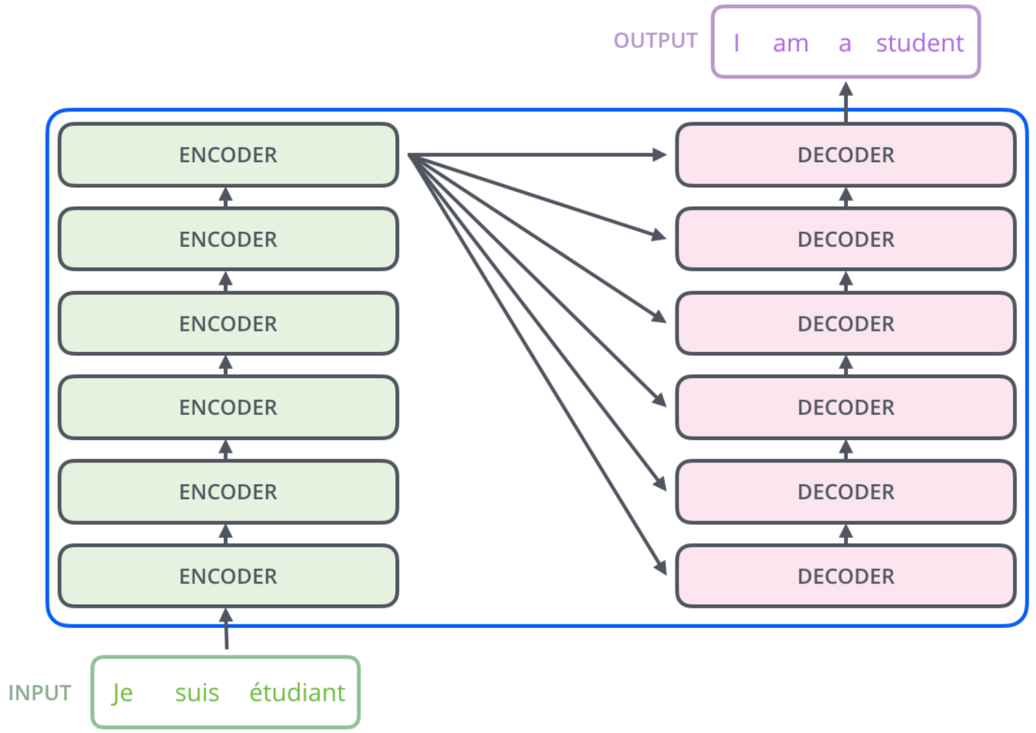
Last time: Attention + RNN in NMT



Transformer

Transformer

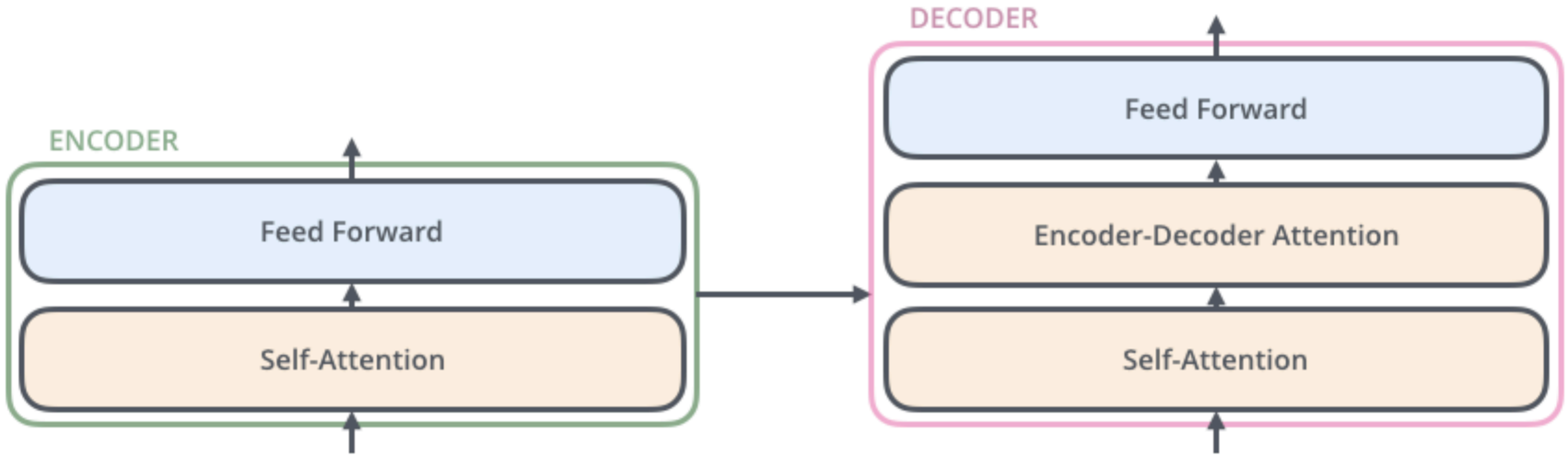
- An architecture that replies entirely on attention without using CNN/RNN
- Proposed in ``Attention Is All You Need'' (Vaswani et al., 2017)
- Initially used for neural machine translation



Transformer

Encoder and Decoder

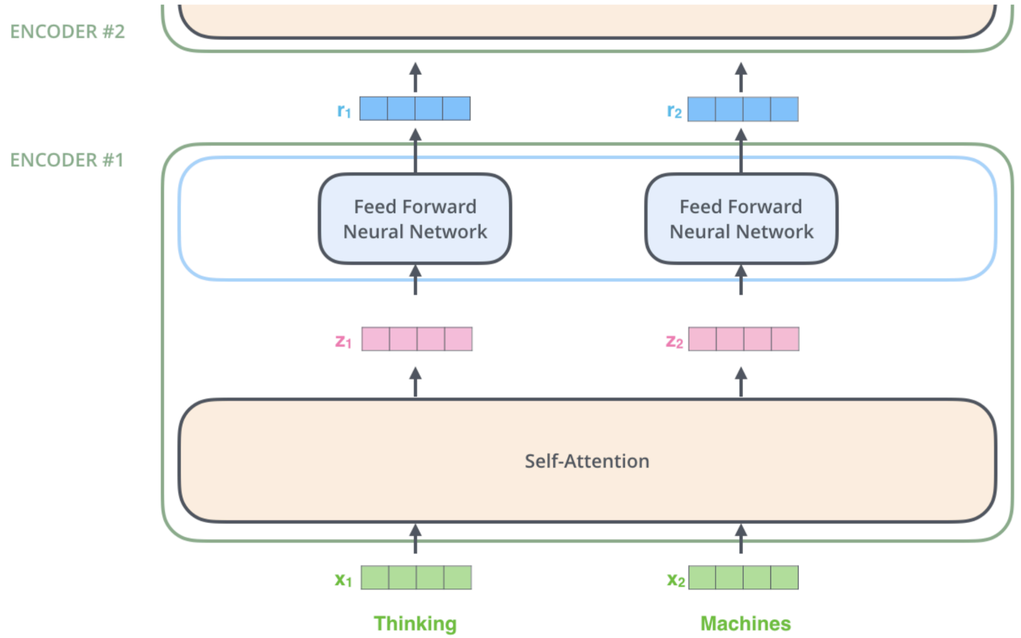
- Self attention layer: the main architecture used in Transformer
- Decoder: will have another attention layer to help it focuses on relevant parts of input sentences.



Transformer

Encoder

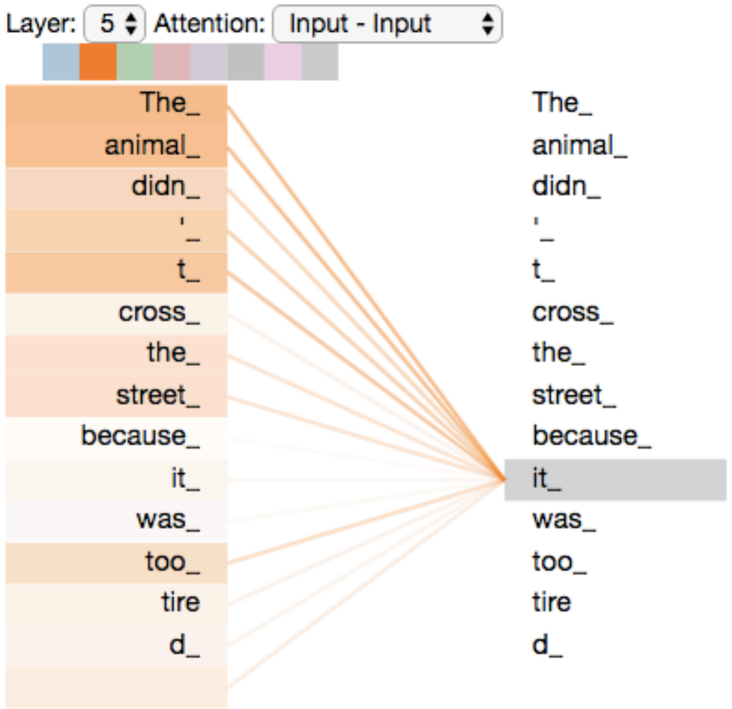
- Each word has a corresponding ``latent vector'' (initially the word embedding for each word)
- Each layer of encoder:
  - Receive a list of vectors as input
  - Passing these vectors to a self-attention layer
  - Then passing them into a feed-foward layer
  - Output a list of vectors



Transformer

Self-attention layer

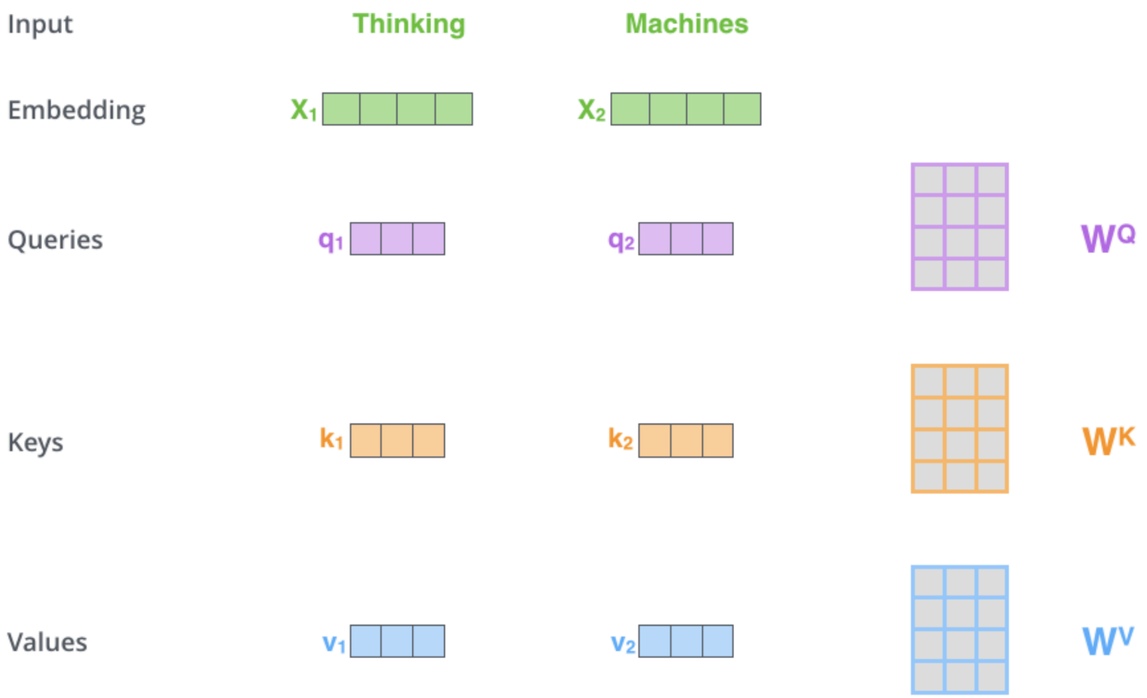
- Main idea: The actual meaning of each word may be related to other words in the sentence
- The actual meaning (latent vector) of each word is a weighted (attention) combination of other words (latent vectors) in the sentences



Transformer

Self-attention layer

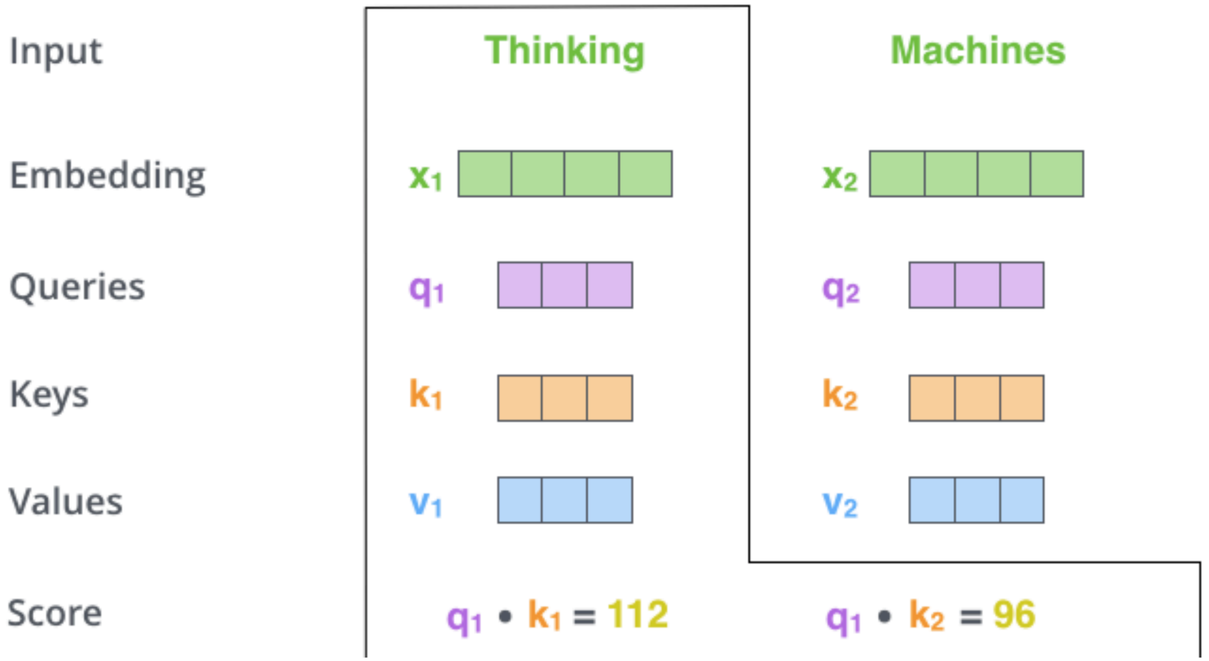
- Input latent vectors: $x_1, \ldots, x_n$
- Self-attention parameters: W^Q, W^K, W^V (weights for query, key, value)
- For each word i, compute
  - Query vector: $q_i = x_i W^Q$
  - Key vector: $k_i = x_i W^K$
  - Value vector: $v_i = x_i W^V$



Transformer

Self-attention layer

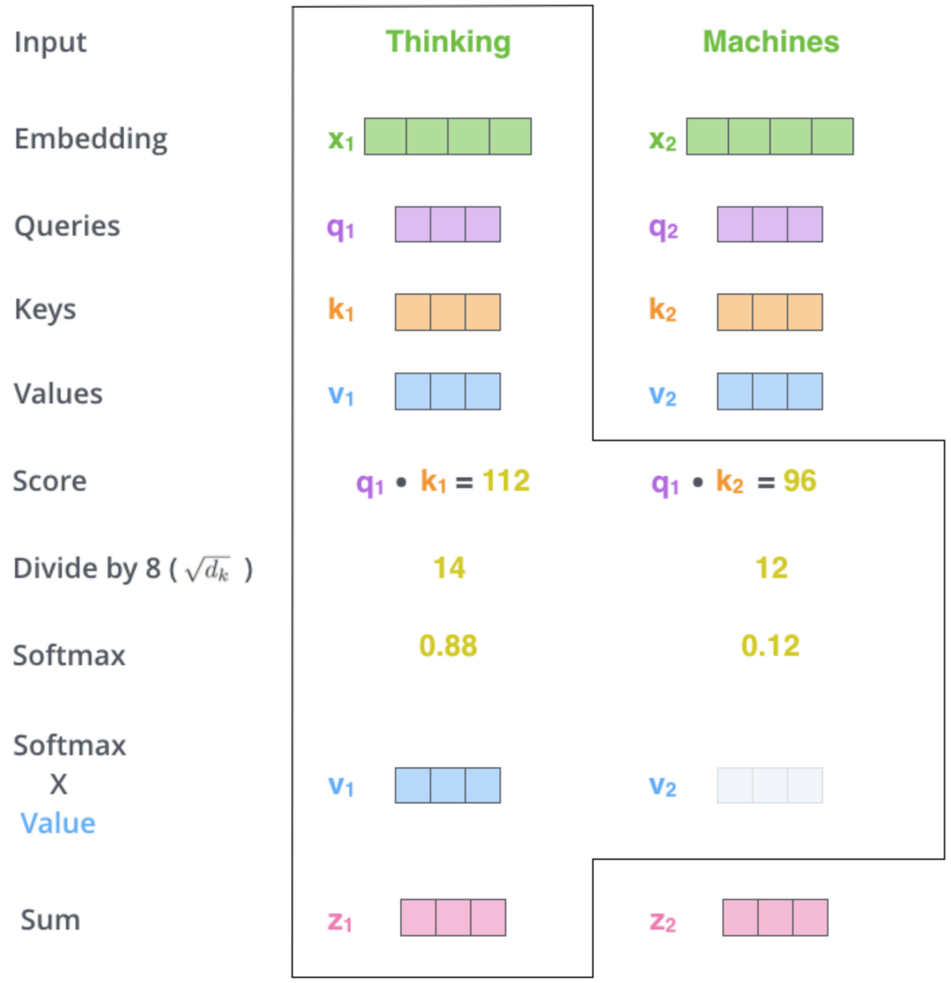
- For each word i, compute the scores to determine how much focus to place on other input words
  - The attention score for word j to word i: $q_i^T k_j$



Transformer

Self-attention layer

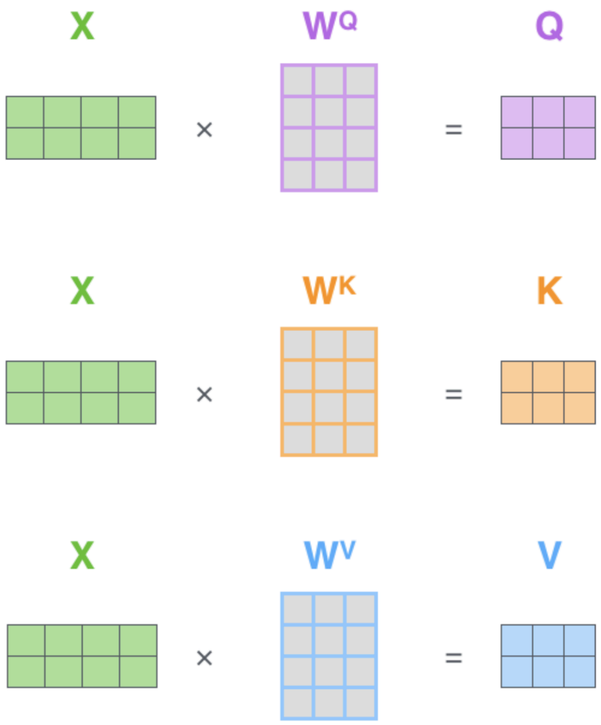
- For each word i, the output vector
  - $\sum_j s_{ij} v_j, \quad s_i = \text{softmax}(q_i^T k_1, \ldots, q_i^T k_n)$

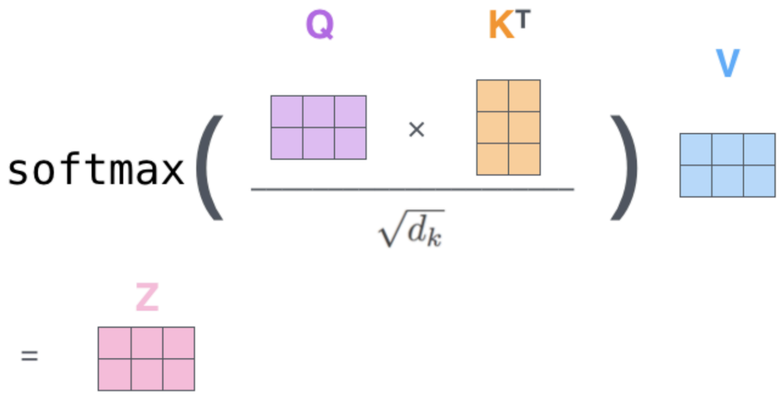


Transformer

Matrix form

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

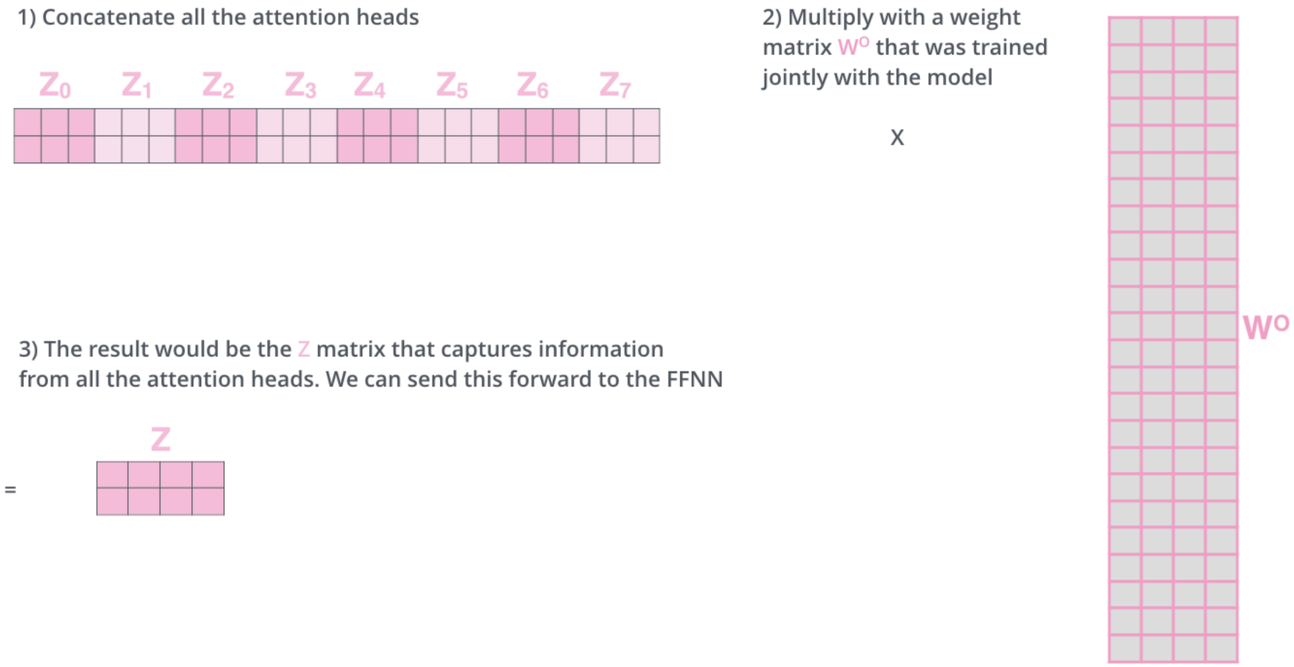




Transformer

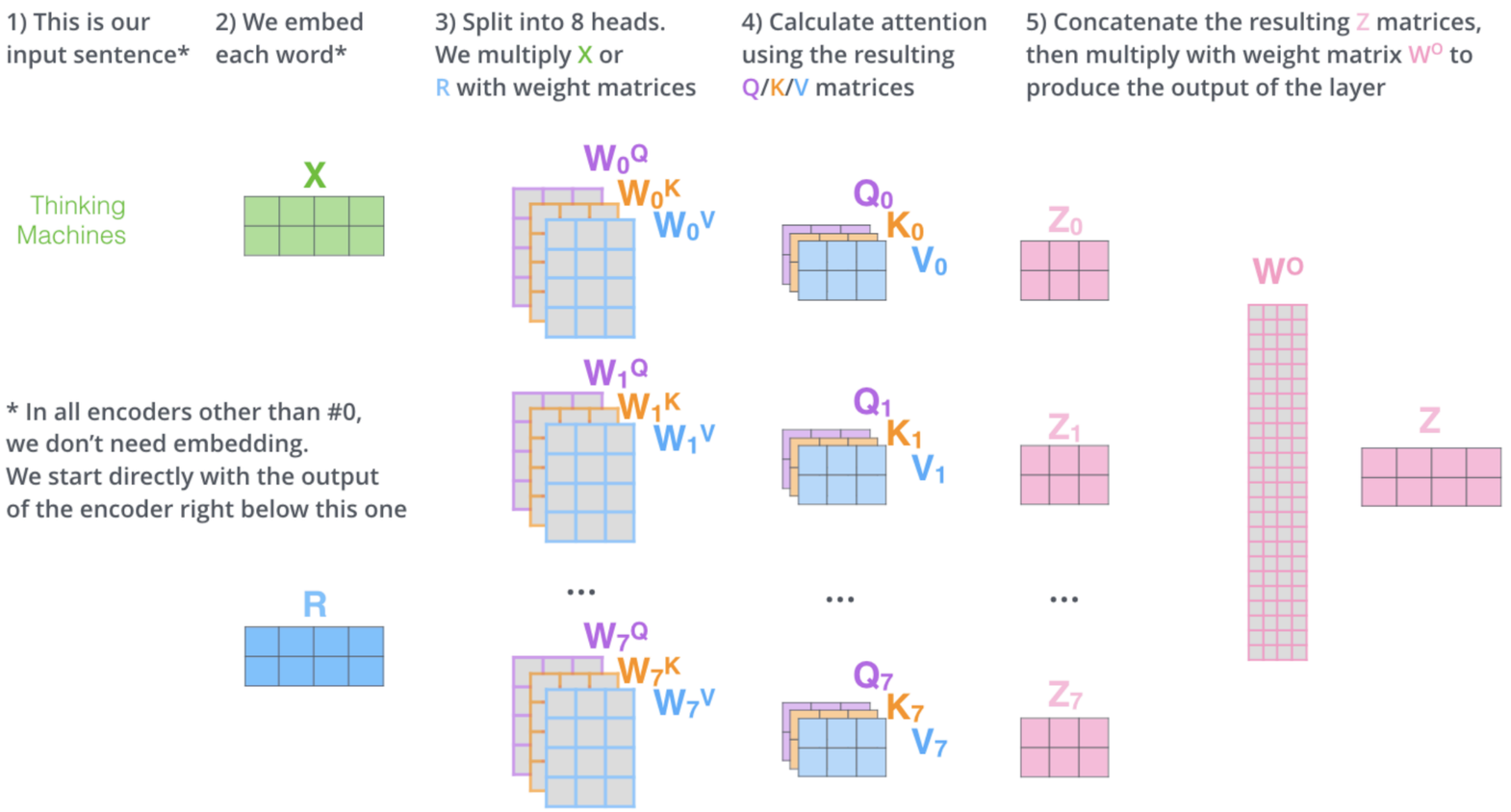
Multiply with weight matrix to reshape

- 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

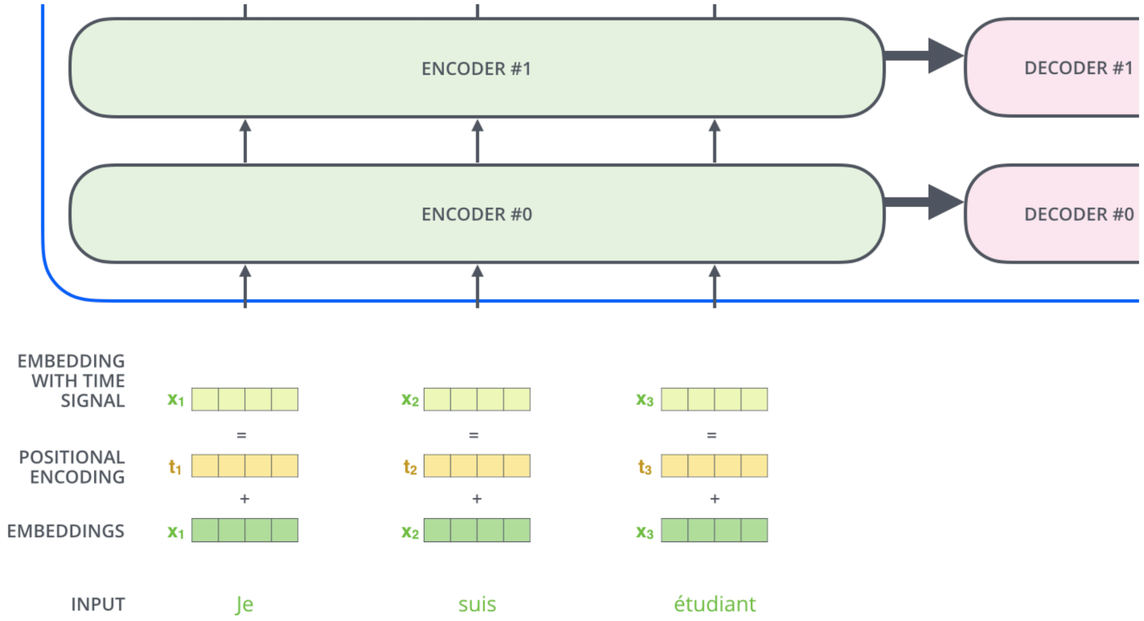
Overall architecture



Transformer

Sinusoidal Position Encoding

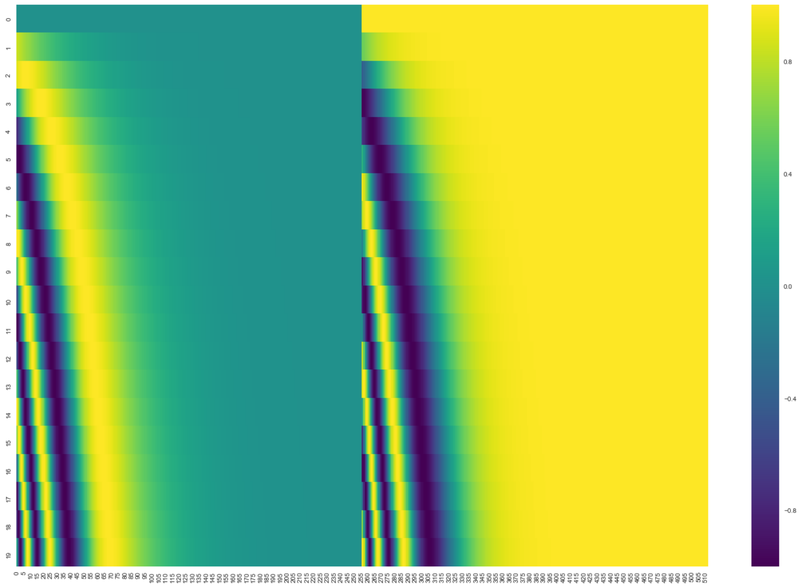
- The above architecture ignores the sequential information
- Add a positional encoding vector to each x_i (according to i)



Transformer

Positional Embedding

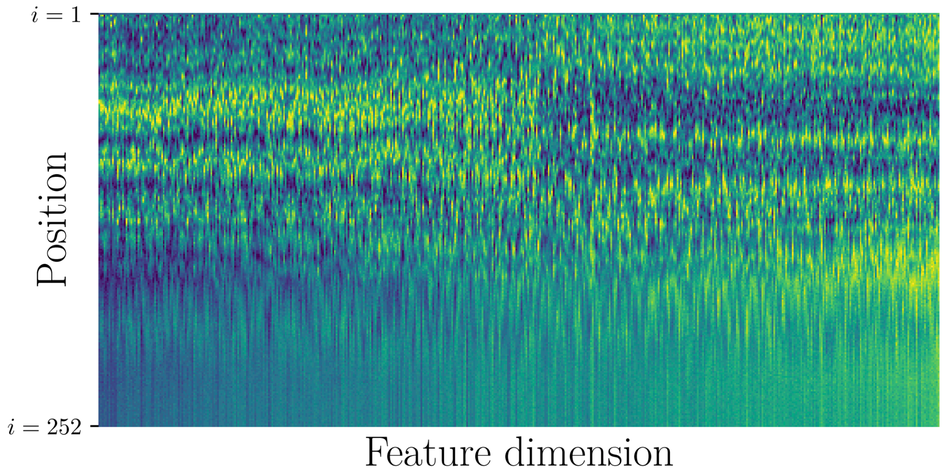
- Sin/cosine functions with different wavelengths (used in the original Transformer)
- The jth dimension of ith token $p_i[j] = \begin{cases} \sin(i \cdot c^{\frac{j}{d}}) \text{ if j is even} \\ \cos(i \cdot c^{\frac{j-1}{d}}) \text{ if j is odd} \end{cases}$
- smooth, parameter-free, inductive



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

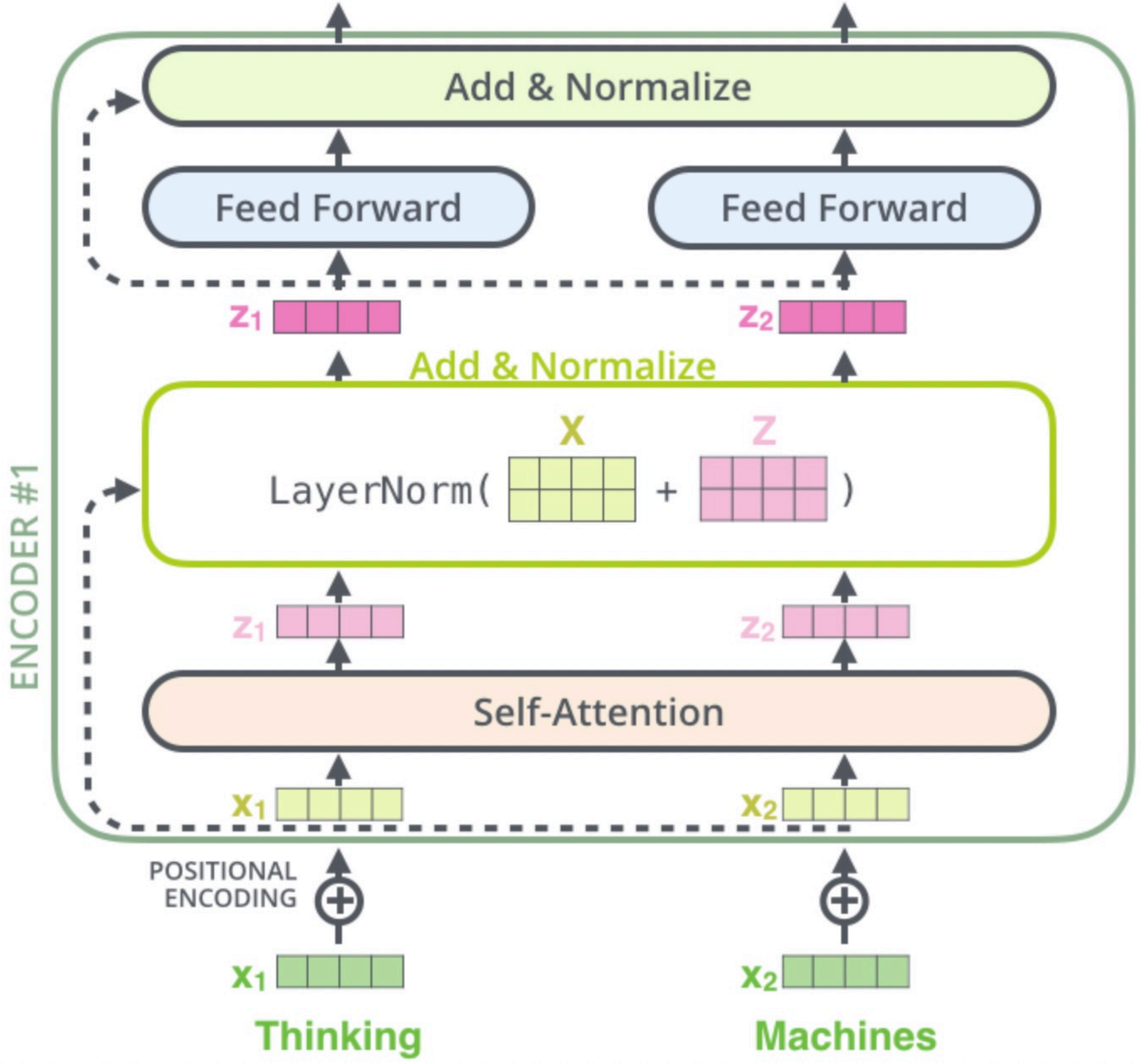


Transformer
Positional Encoding

- 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

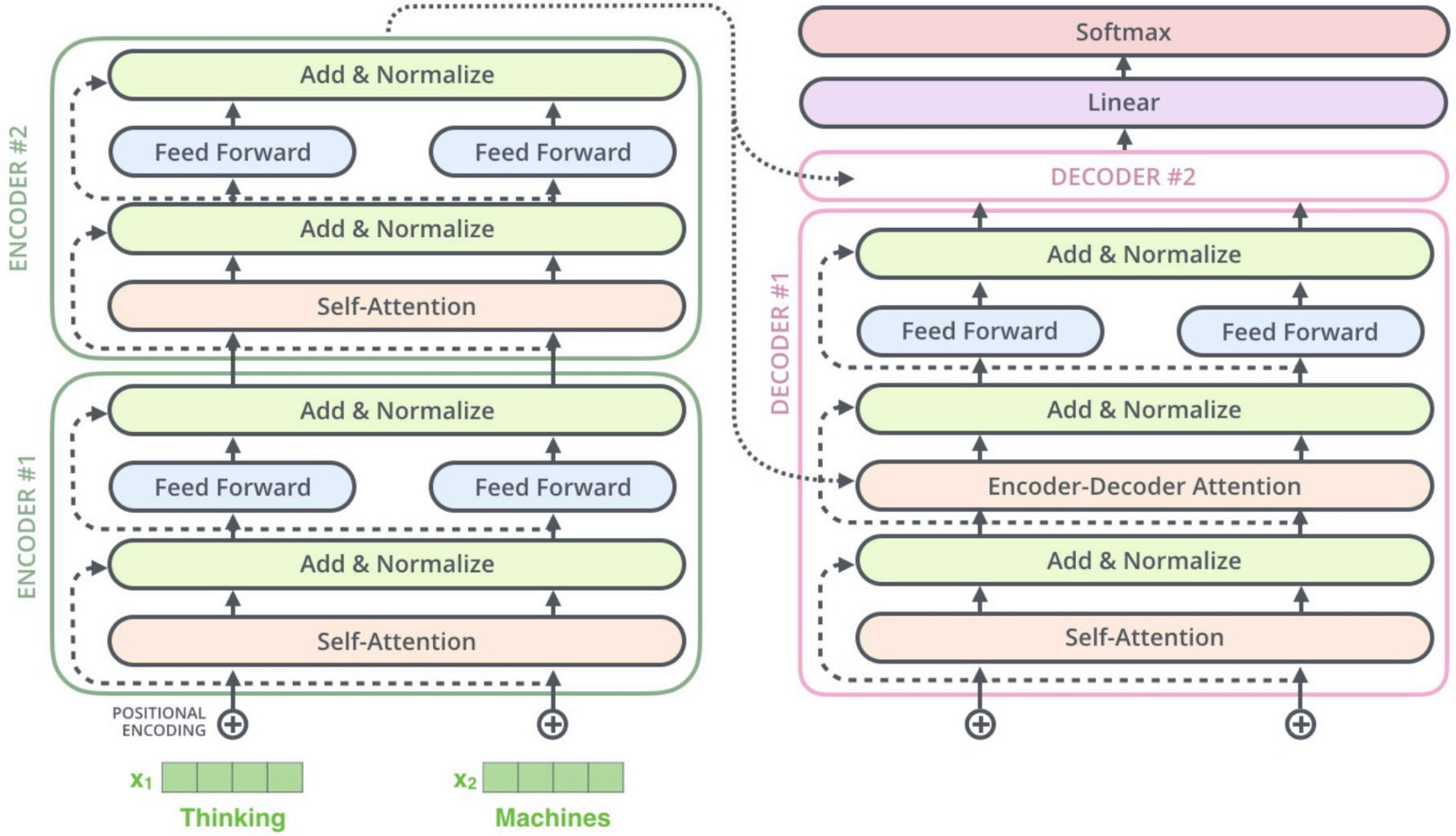
Transformer

Residual



Transformer

Whole framework



Unsupervised pertaining for NLP

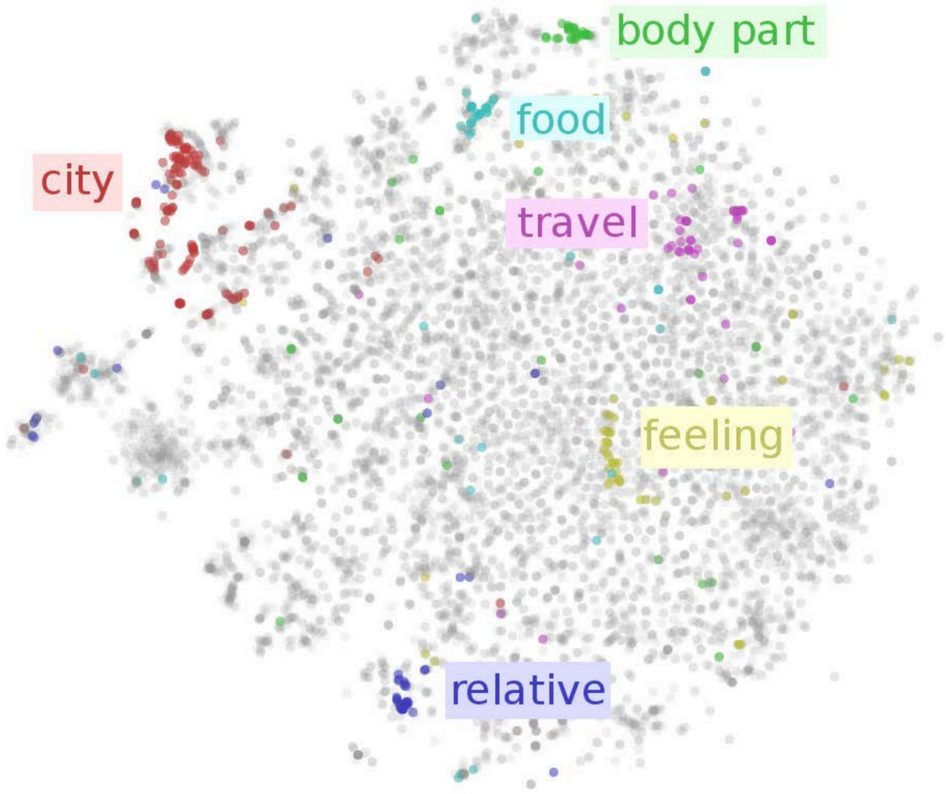
Motivation

- 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