Image/Text Representation Learning

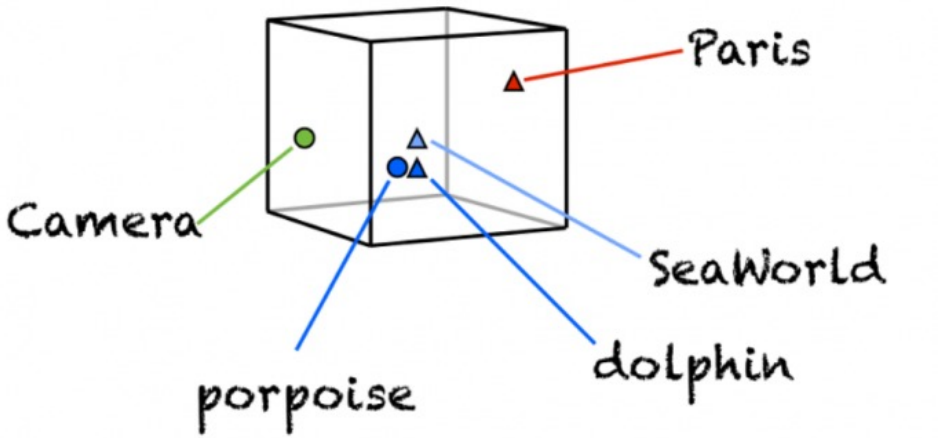
Qinliang Su （苏勤亮）

Sun Yat-sen University

suqliang@mail.sysu.edu.cn

What is representation learning?

In a nutshell, representation learning is to map each data instance x_i to a vector in space $\mathbb{R}^d$



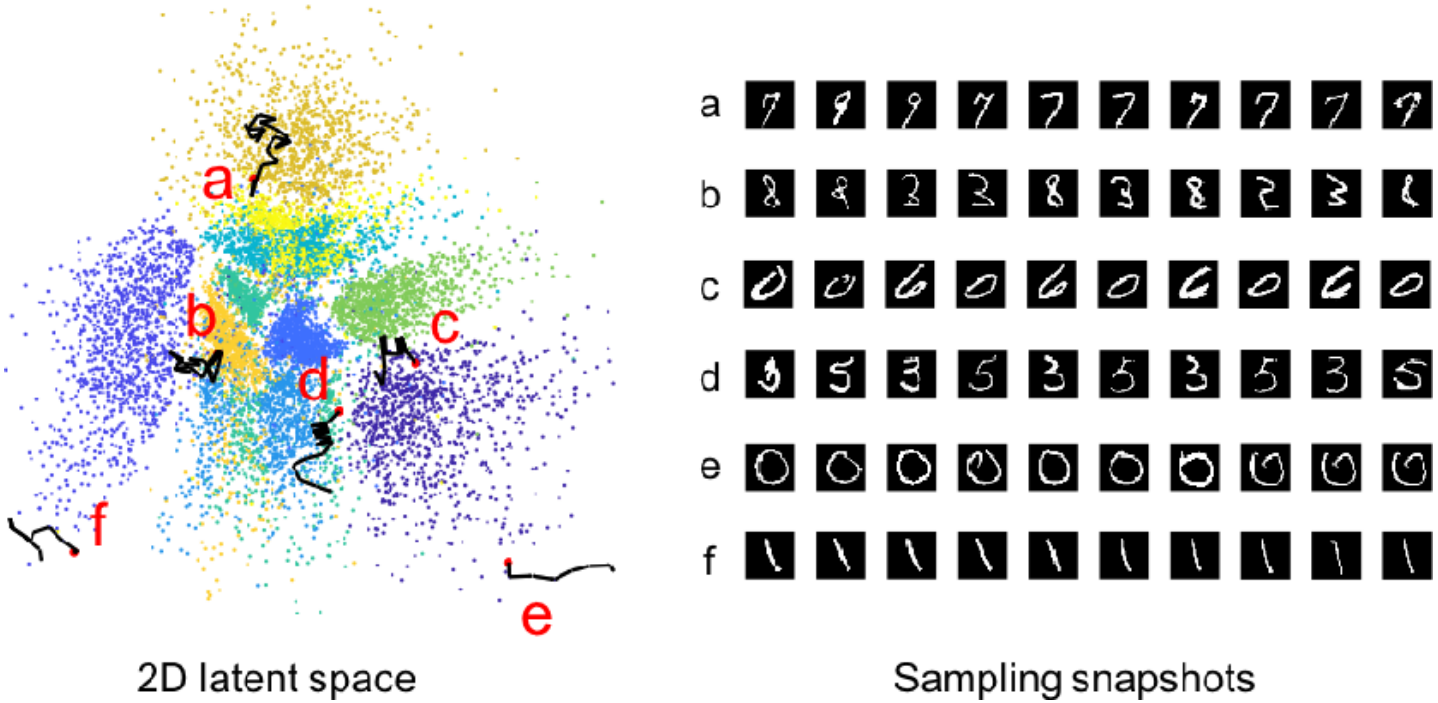
- The data x_i could be of any format, *e.g.*, images, words, sentences, videos, voices etc.

Obviously, the mapping is not unique. Data could be mapped to vectors in countless ways

- What constitute a good representation?

Representations that *preserve as much as possible the semantic information of original data, instead of the whole information*

Example: By representing hand-written digits with vectors in $\mathbb{R}^2$, for good representations, the representations of images from the same digit should be close to each other



2D latent space

Sampling snapshots

Why is representation learning needed?

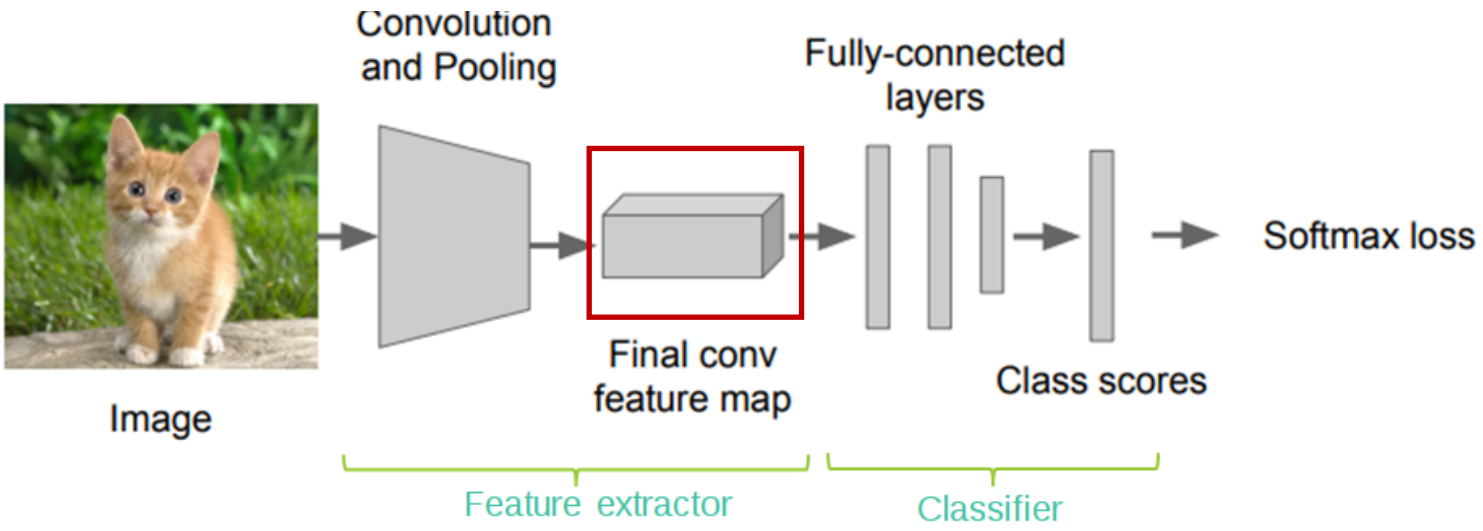
- Comparing with raw data, representation is generally more friendly to downstream applications. This is because representations are
  - Much more compact
  - Discarding irrelevant information, but preserving the most important *semantic information*

- Representations could be applied to various kinds of downstream applications, such as
  - Classification
  - Clustering
  - Similarity search
  - Anomaly detection

Image Representation

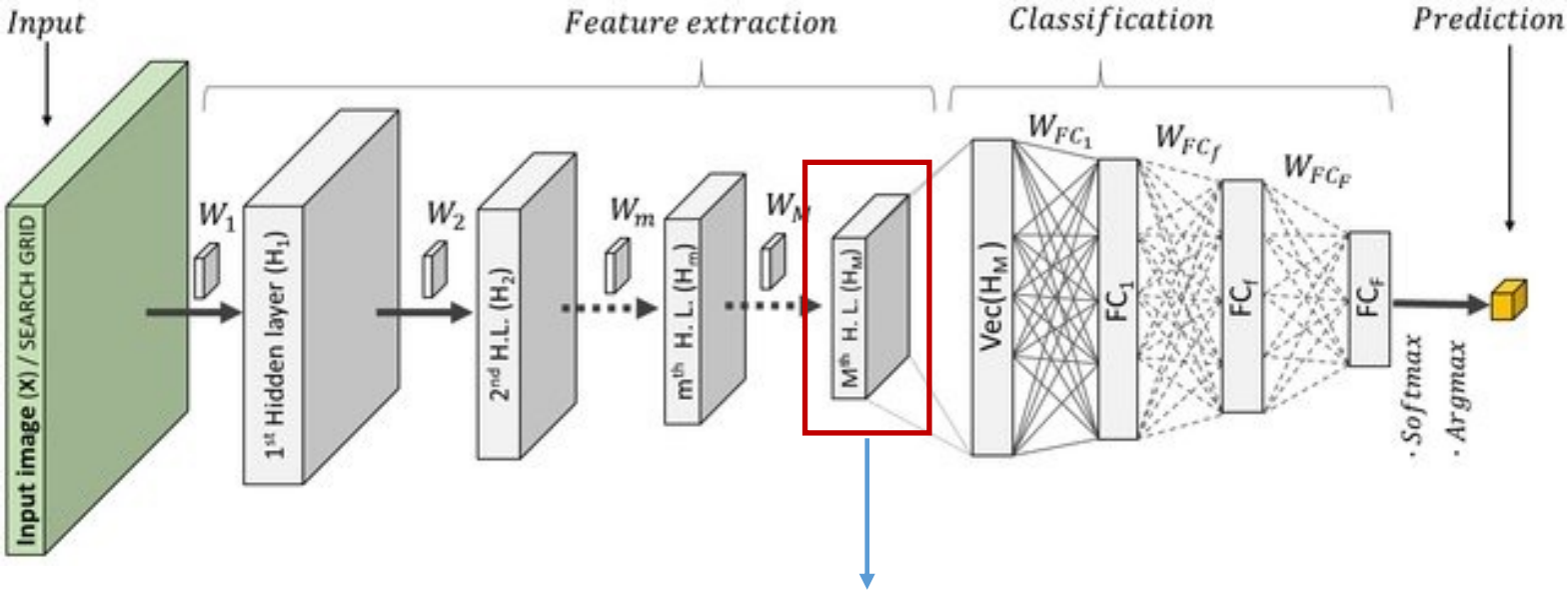
Supervised Case

- Given a set of image-label pairs $\{x_i, y_i\}_{i=1}^{N}$, we train a classifier on the given image-label pairs



- Feature maps output the last convolutional layer can be viewed as the representations of input images

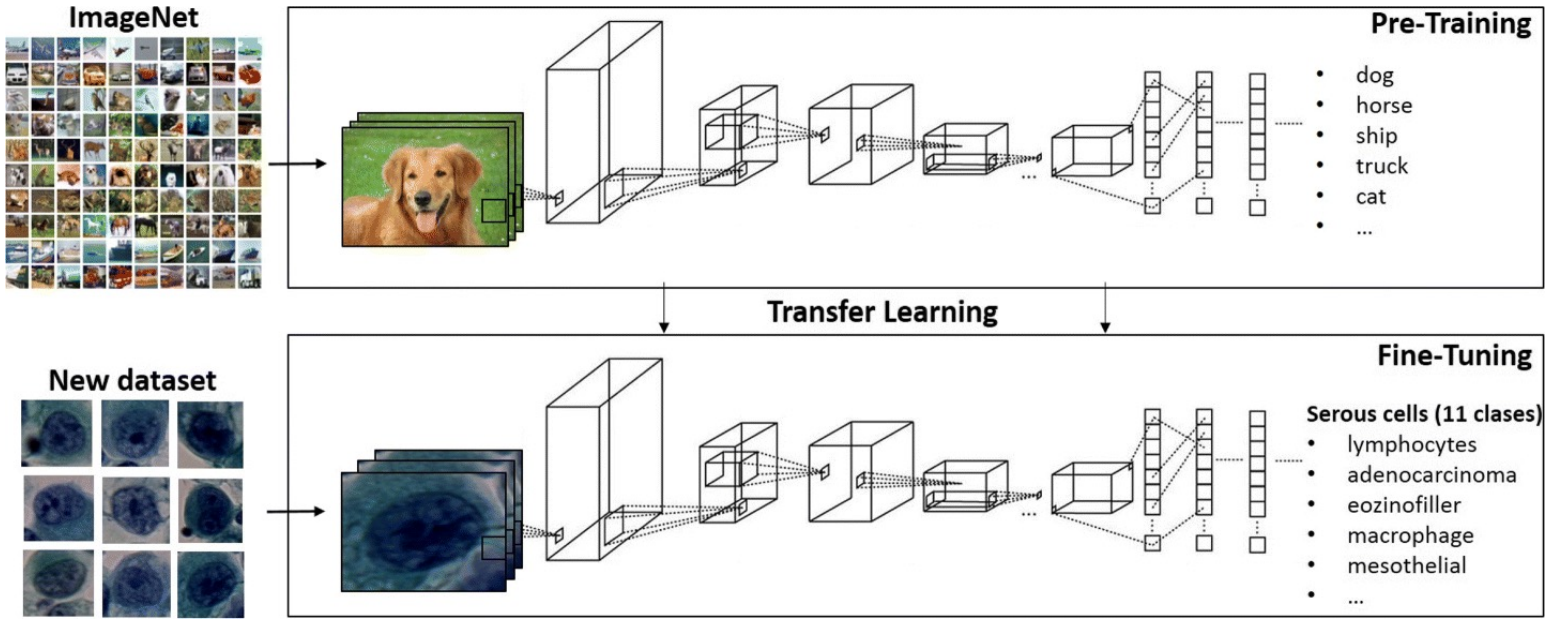
Why??



In addition to convolutional feature maps, *outputs from the fully-connected penultimate layer* are also sometimes used as the representations

Without Labels

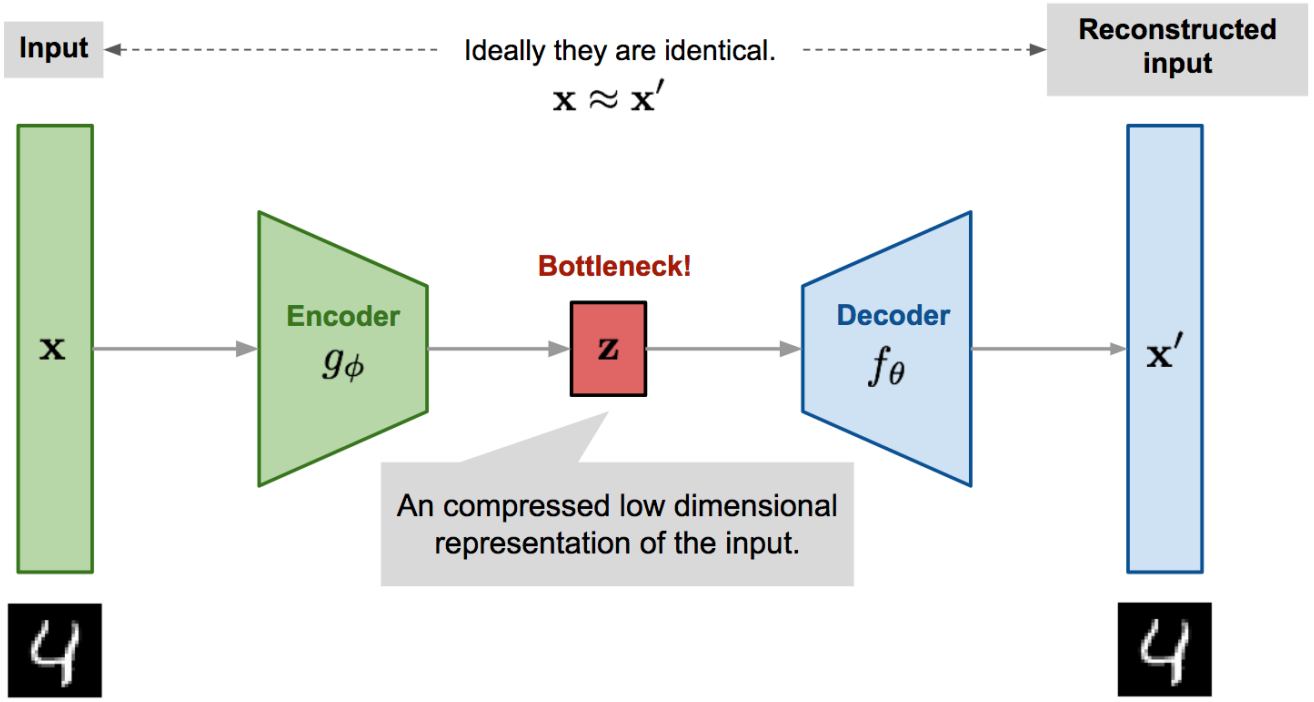
- When the given dataset does not include labels,
  1) Training a CNN on an *auxiliary labeled dataset* (*e.g.* ImageNet)
  2) Representations of images from a new dataset can be obtained by passing images through the pre-trained CNN



The pre-trained CNN can be viewed as a *feature extractor*

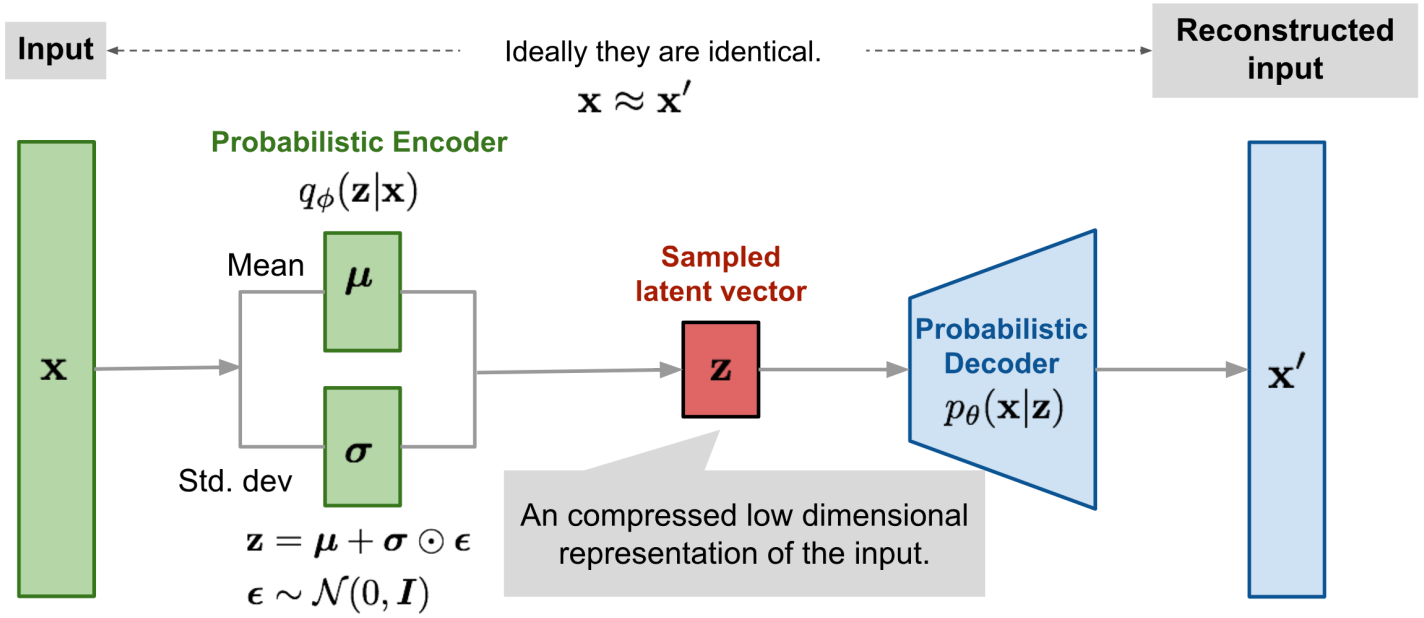
Without Labels & Auxiliary Data

- Under this case, we resort to unsupervised learning
- A direct way is to train an auto-encoder on the given unlabeled data $\{x_i\}_{i=1}^N$, and then use the encoder to extract the representations



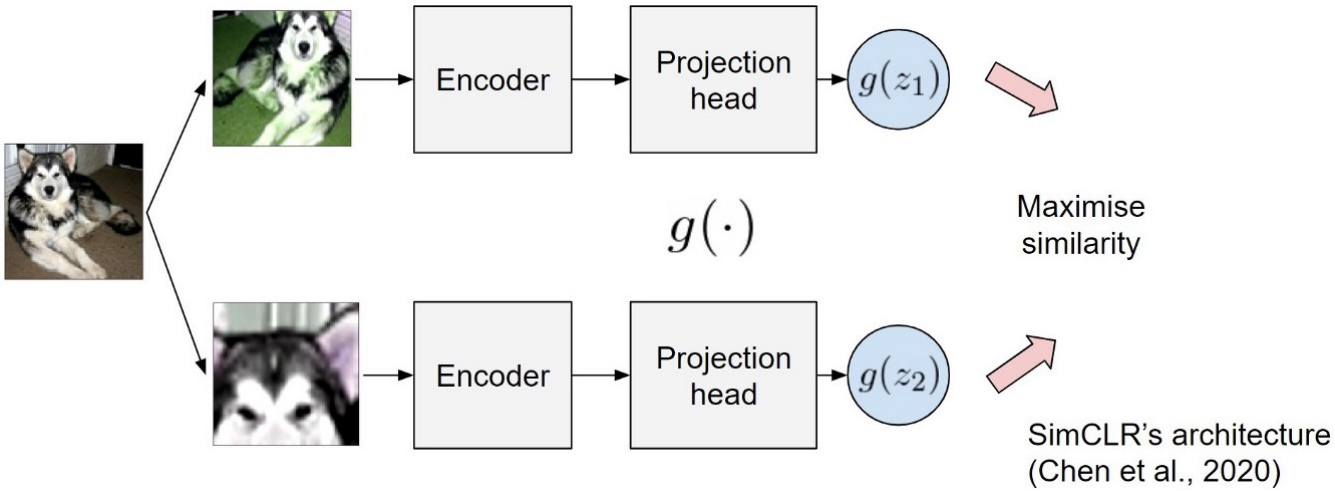
PCA can be seen as a linear auto-encoder

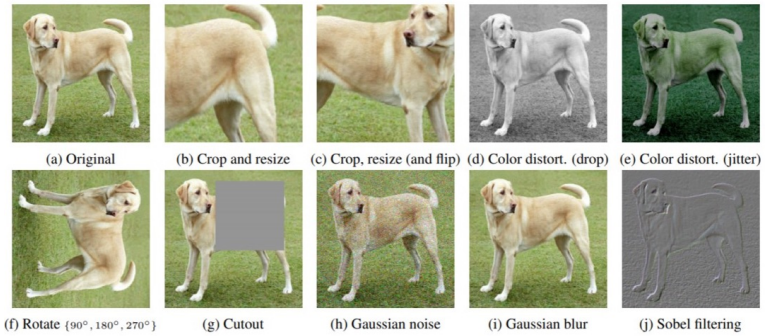
- We can also train generative models on the unlabeled data $\{x_i\}_{i=1}^N$, and use the generative models to extract representations



Using the generative approach, various kinds of prior knowledges can be imposed onto the learning process easily

Contrastive Learning (Unsupervised)





Maximize similarity is to maximize

$$L = \log \frac{e^{sim\left(h_1^{(k)}, h_2^{(k)}\right)/\tau}}{e^{sim\left(h_1^{(k)}, h_2^{(k)}\right)/\tau} + \sum_{n \neq k} e^{sim\left(h_1^{(k)}, h_i^{(n)}\right)/\tau}}$$

- Yield representations comparable to the supervised methods

Text Representation

What is Text Representation?

- Unlike images, natural language (textual data) does not appear in numerical format by nature

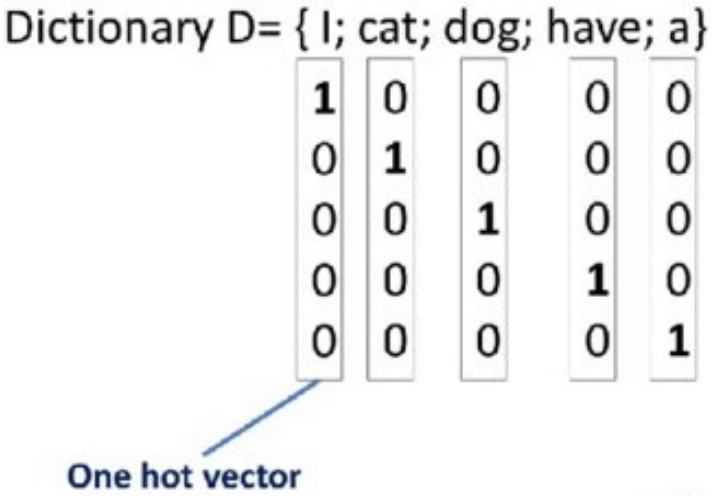
"Today is a good day"

- To learn from textual data, the first thing that we need to do is to transform the textual data into a form that can be processed by computers

Outline

- One-Hot Representation
- Word2Vec Representation
- BERT-Based Representation

- The simplest way to represent textual data is to encode every word in the vocabulary set by a one-hot vector, *e.g.*,



- After that, sentences are represented as the concatenation of the one-hot vectors

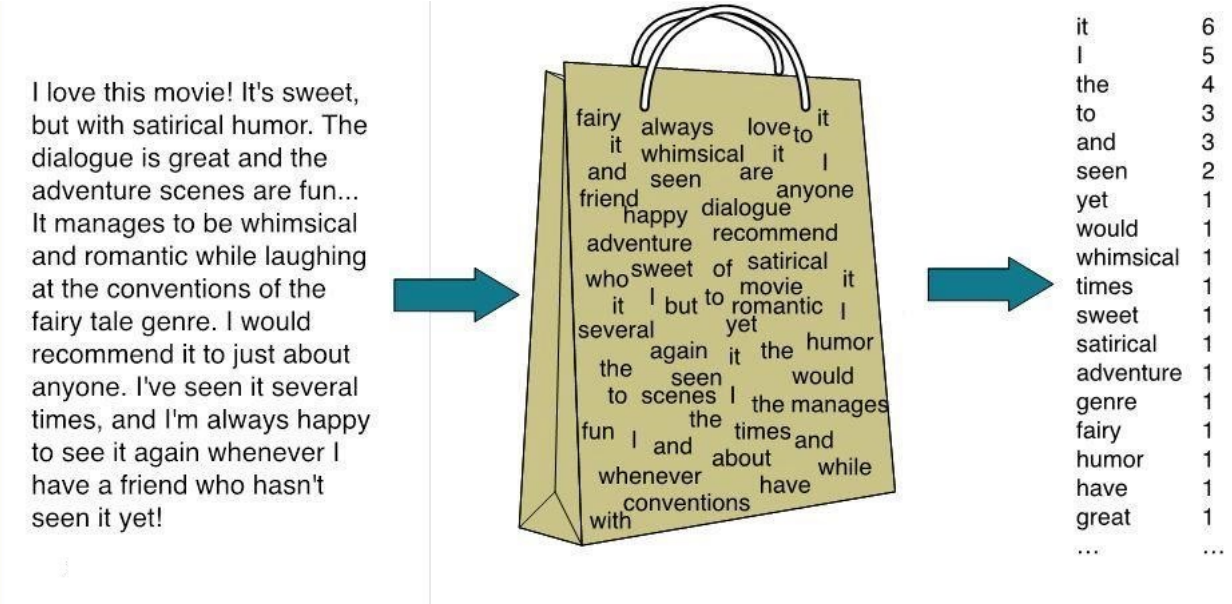
For instance, “I have a dog” can be represented as a matrix on the right

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

➢ Representing sentences as a matrix is cumbersome, especially when the sentences are long

Bag-of-Words (BOW)

- Bag-of-Word (BOW) discards the word order information, only recording the count of each word in a document



- Cons: Each word is considered as equal importance. However, *the amount of information conveyed by different words is not equal*

  For example, words like ‘the’, ‘a’ are much less informative than words like ‘classroom’, ‘football’ *etc*.

TFIDF

- Term Frequency–Inverse Document Frequency (TFIDF) attempts to reflect the unequal importance of different words
- Specifically, it is computed as the product of two statistics, *i.e.*,

$$tfidf(w, d, \mathcal{D}) = tf(w, d) \times idf(w, \mathcal{D})$$

  - w denote the word; d denotes the d-th document, $\mathcal{D}$ denotes the corpus (a collection of documents)
  - $tf(w, d)$ denotes the frequency of word w in document d

$$tf(w, d) = \frac{\#\ of\ word\ w\ in\ d}{\#\ of\ all\ words\ in\ d}$$

  - $idf(w, \mathcal{D})$ the log inverse fraction of the documents containing word w in the corpus $\mathcal{D}$

$$idf(w, \mathcal{D}) = \log \frac{\#\ of\ documents\ in\ \mathcal{D}}{\#\ of\ documents\ containing\ word\ w\ in\ \mathcal{D}}$$

- $idf(w, \mathcal{D})$ measures how much information the word w contains. The larger the value $idf(w, \mathcal{D})$ is, the more informative the word w is

- Thus, TFIDF feature $tfidf(w, d, \mathcal{D}) = tf(w, d) \times idf(w, \mathcal{D})$ accounts for both of a word's frequency in a document and its informativeness in the corpus

$$\mathrm{tfidf}(t, d, D) = \mathrm{tf}(t, d) \cdot \mathrm{idf}(t, D)$$

		blue	bright	can	see	shining	sky	sun	today
Document 1	1	**0.301**	0	0	0	0	0.151	0	0
Document 2	2	0	0.0417	0	0	0	0	0.0417	**0.201**
Document 3	3	0	0.0417	0	0	0	**0.100**	0.0417	0
Document 4	4	0	0.0209	**0.100**	**0.100**	**0.100**	0	0.0417	0

Each row means the TFIDF feature of a document

Limitations

- One-hot word representations
  - Unable to reflect the *semantic similarities* of words, *e.g.*, 'football' and 'soccer'
  - Too cumbersome when representing a document
  - The dimension is very *high*, as large as 20000
  - It is very *sparse*
- The BOW and TFIDF document representations
  - Unable to reflect the *semantic similarities* of words
  - Do not contain any *word-order information*
  - The dimension is very *high*
  - It is very *sparse*

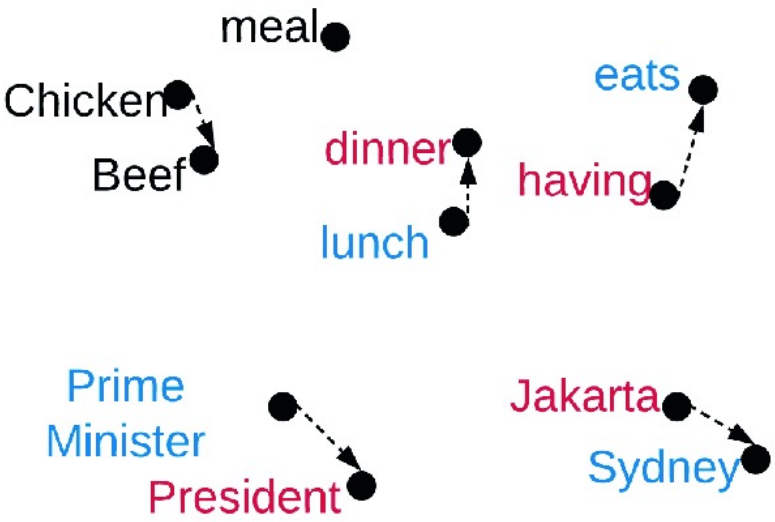
} But much better than one-hot representatons

Outline

- One-hot Representation
- Word2Vec Representation
- BERT-Based Representation

Word2Vec Word Embedding

- Goal: Represent words by real-valued vectors which have the characteristics
  1) being *dense* (low dimensional)
  2) reflecting the *semantic similarities* among words



- Sentence embedding can be constructed from word embedding subsequently

How to Learn Word2Vec Embedding?

- Observation: Semantic similarities of words are implicitly reflected in existing sentences

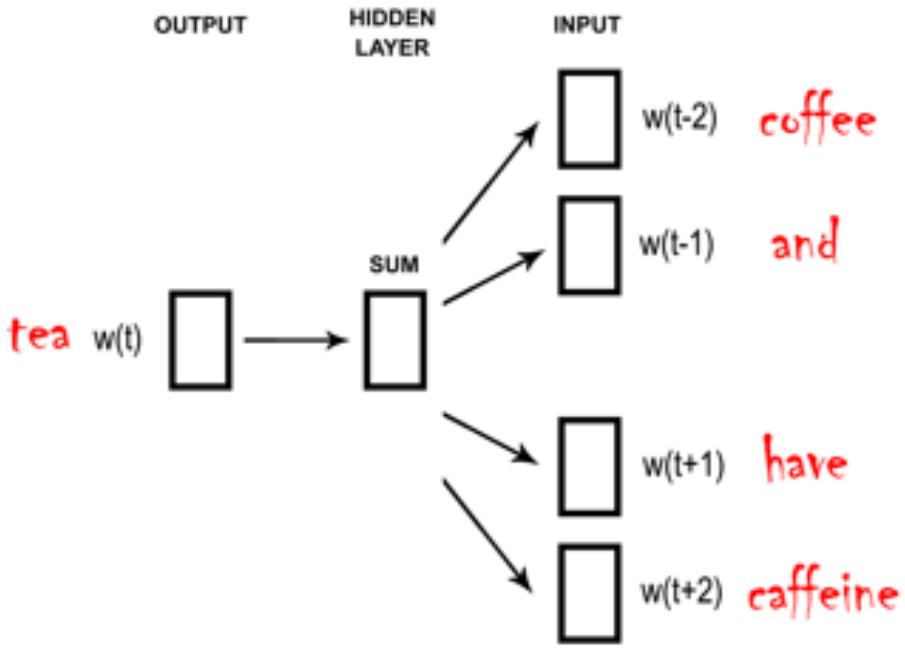
  For instance, if two words often appear simultaneously, they may preserve similar meaning

A cup of **tea**
A cup of **coffee**
Tea or **coffee**?
Coffee and **tea** have caffeine
Let's go for a **coffee**
Let's get a **tea**
Coffee vs **Tea**: Which is Best?
I avoid adding sugar to my **tea**
I drink **coffee** with two spoons of sugar

- Thus, *semantics-preserving* embedding could be learned from the huge amount of sentences existing in the world

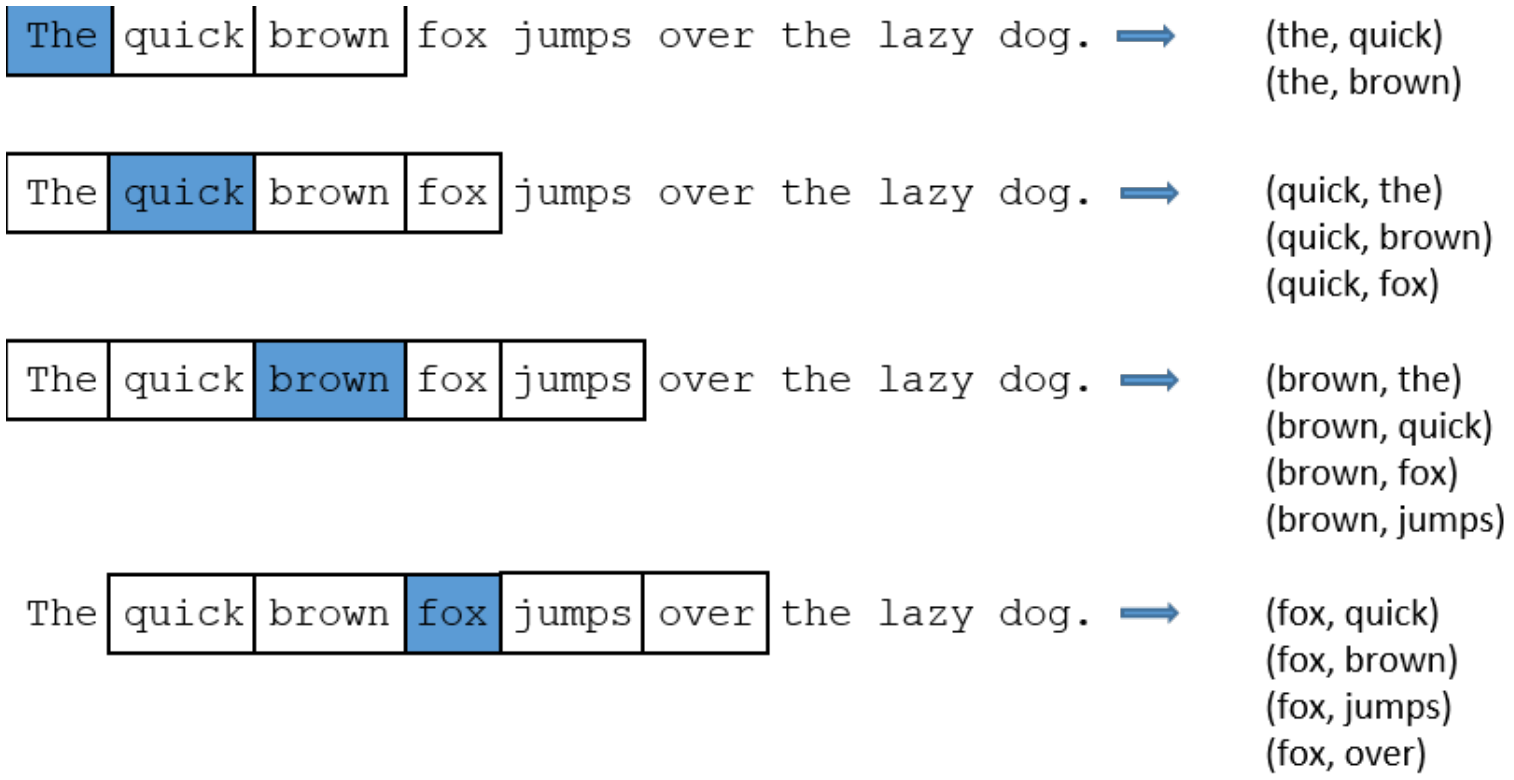
Skip-gram

- Two basic methods
  1) Skip-grams
  2) Continuous Bag-of-Words (CBOW)
- Skip-gram: predicting the context using the center word



- Initialize the embedding of each word by a random vector $u \in \mathbb{R}^m$
- Then, using the t-th word w_t to predict its left and right words. The objective function is

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m \le j \le m \\ j \neq 0}} \log P\left(w_{t+j} \mid w_t; \theta\right)$$

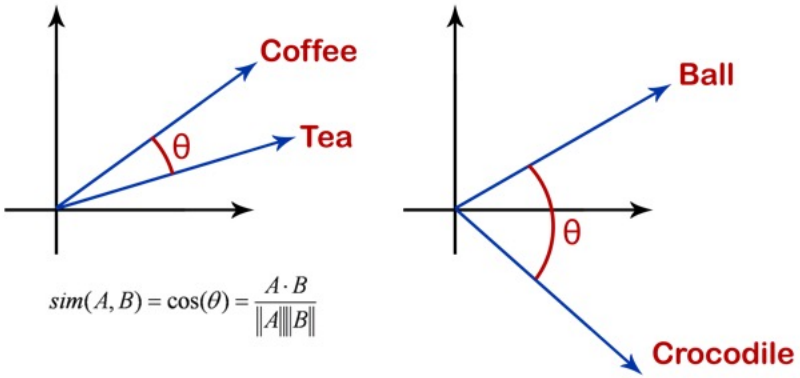


- The prediction probability $P(w_{t+j} \mid w_t; \theta)$ is modeled as

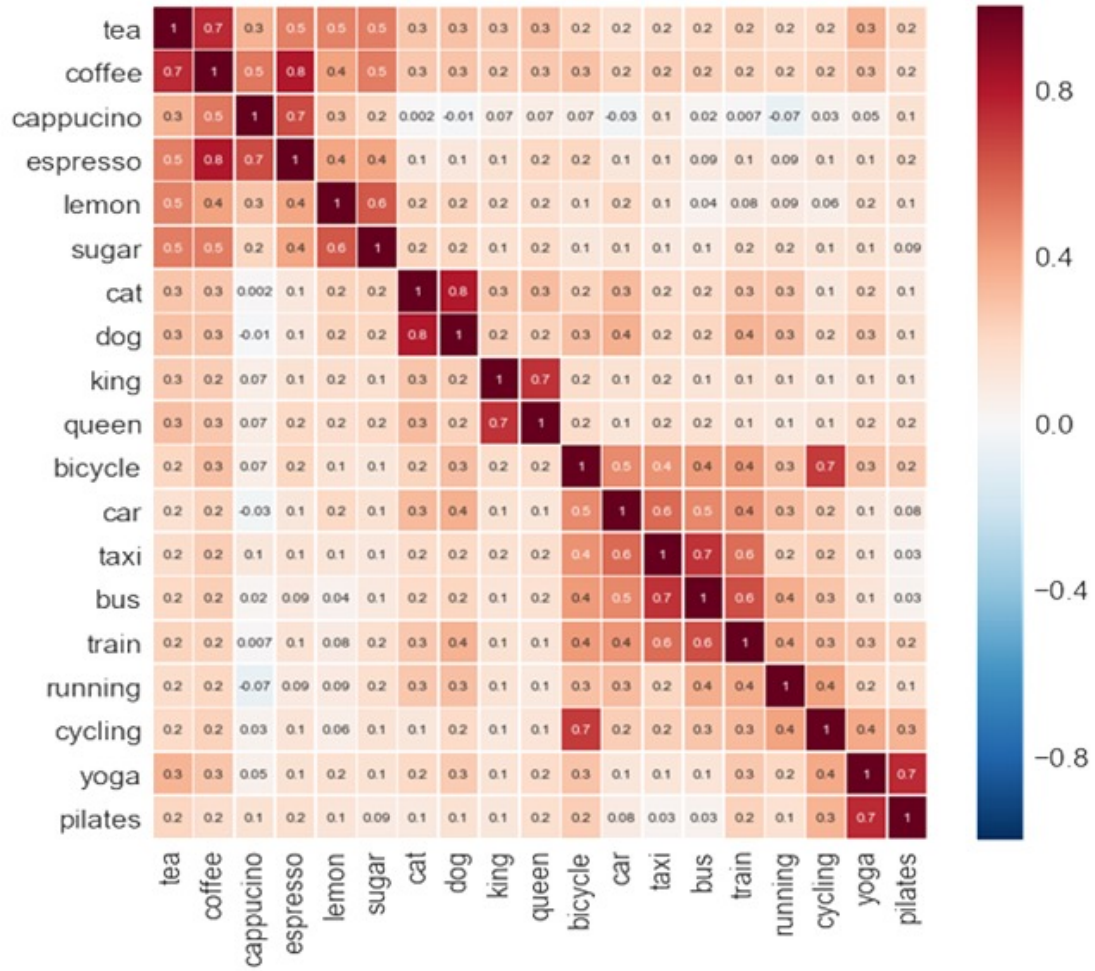
$$P(w_{t+j} \mid w_t; \theta) = \frac{\exp\left(u_{w_t}^T u_{w_{t+j}}\right)}{\sum_{w \in \mathcal{V}} \exp\left(u_{w_t}^T u_w\right)}$$

  - $\mathcal{V}$ is the set of vocabulary words
  - u_w denotes the embedding of word w, which is to be optimized

- After training on a huge corpus, *semantic similarities of words are reflected in the cosine similarities of their embeddings*

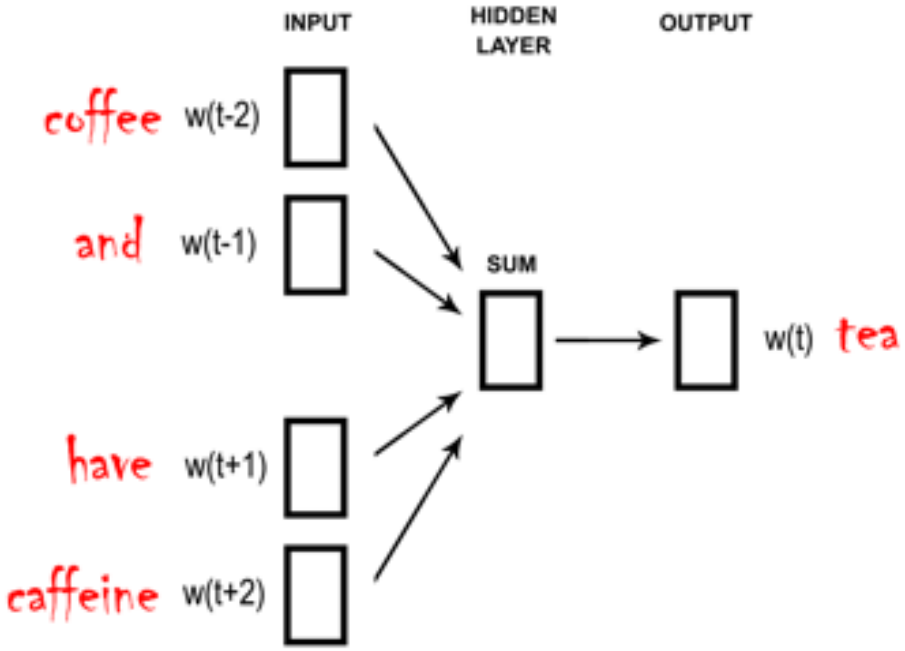


- Example of cosine similarities evaluated on word embeddings that are trained on a large corpus



Continuous Bag-of-Words

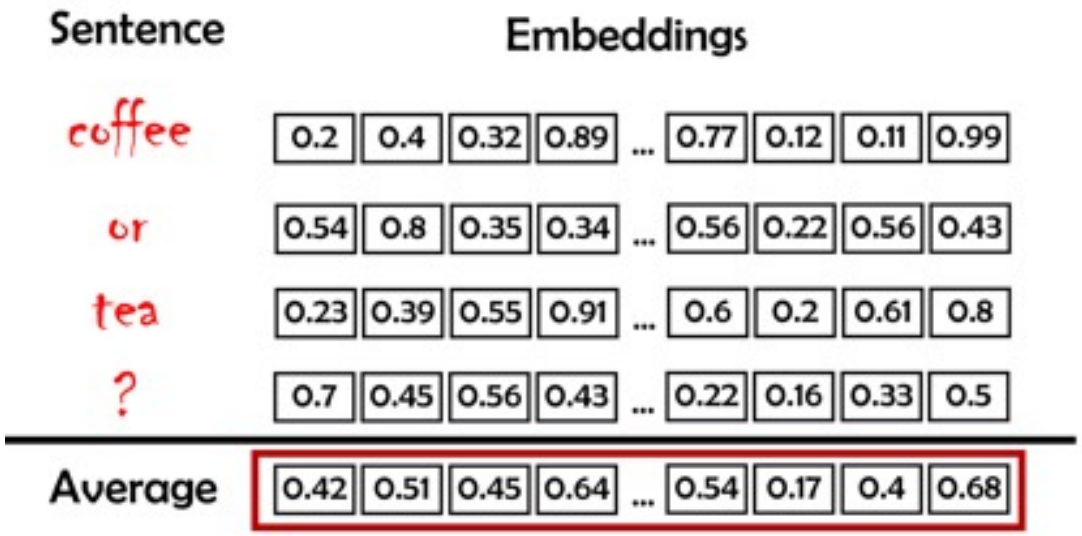
- Different from the skip-grams, CBOW predicts the center word using left and right words



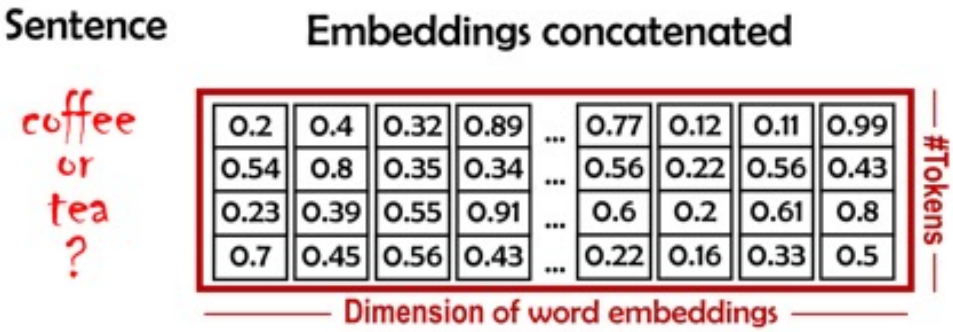
Sentence Embedding

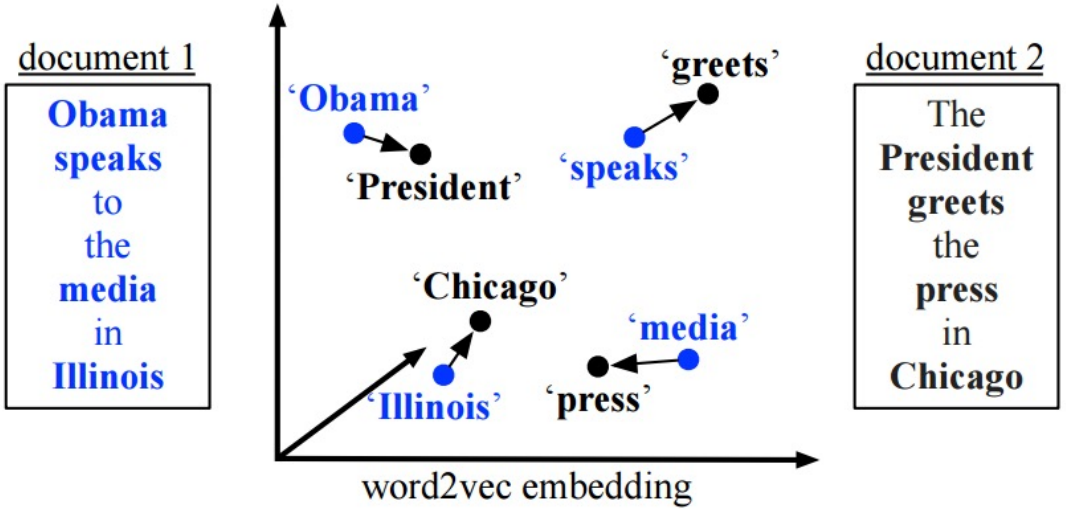
- Sentence embedding can be obtained from word emebddings in many different ways

1) Average

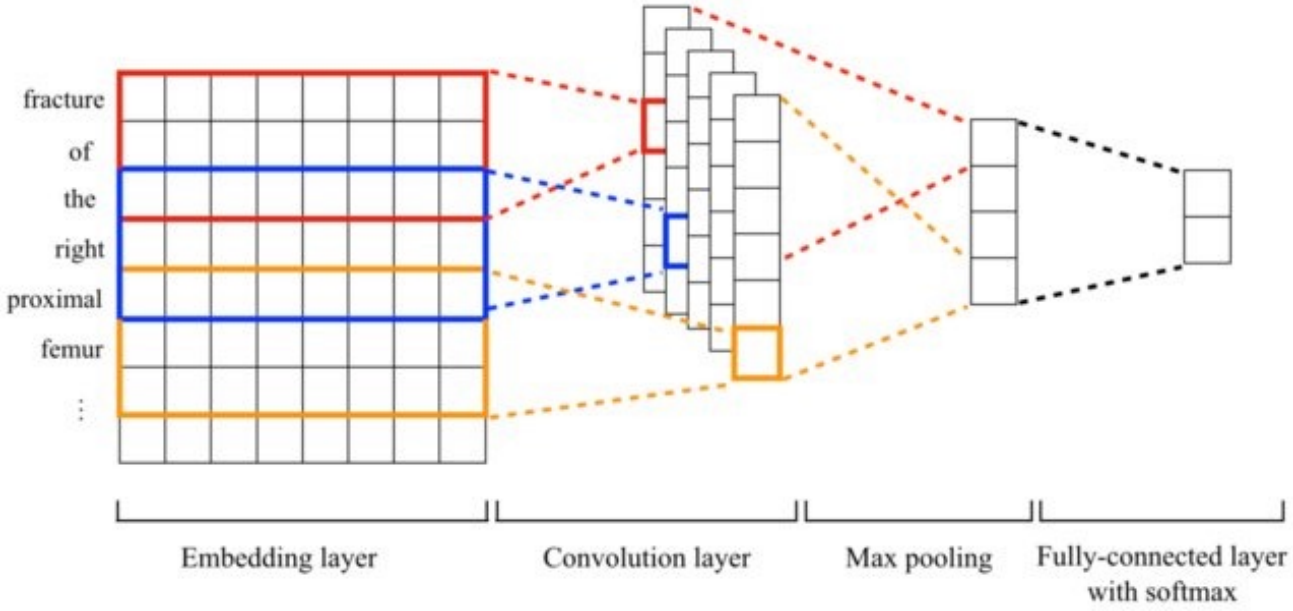


2) Concatenation



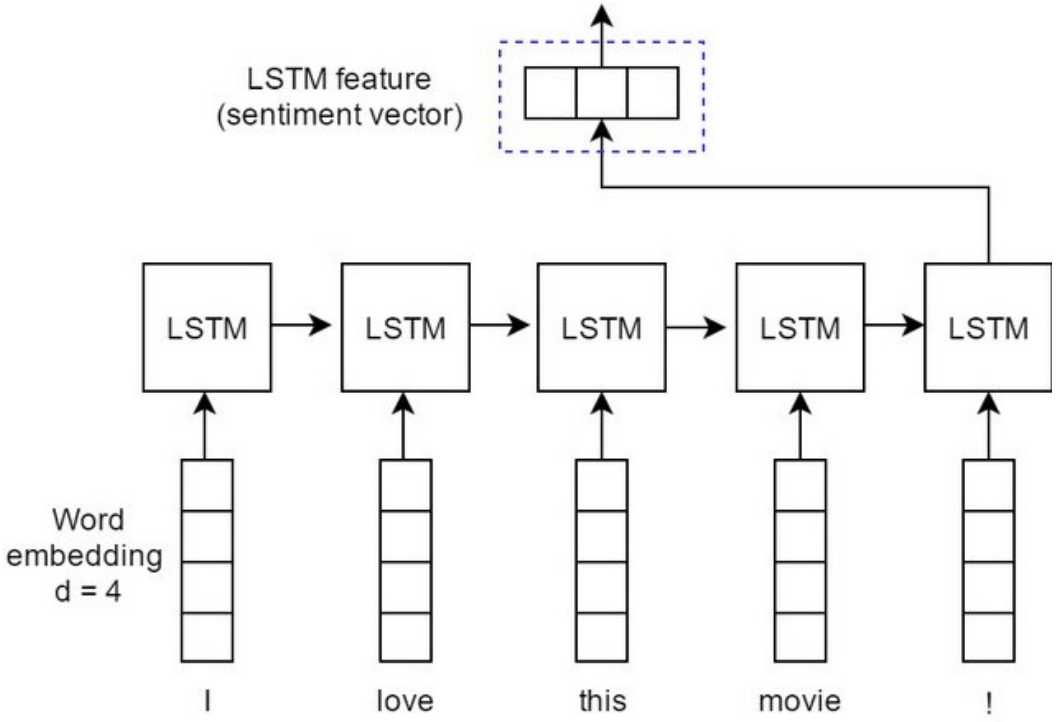


3) Extracting sentence embedding with CNNs (Text CNN)



The parameters of CNN are optimized using a downstream task

4) Extracting sentence embedding with RNN

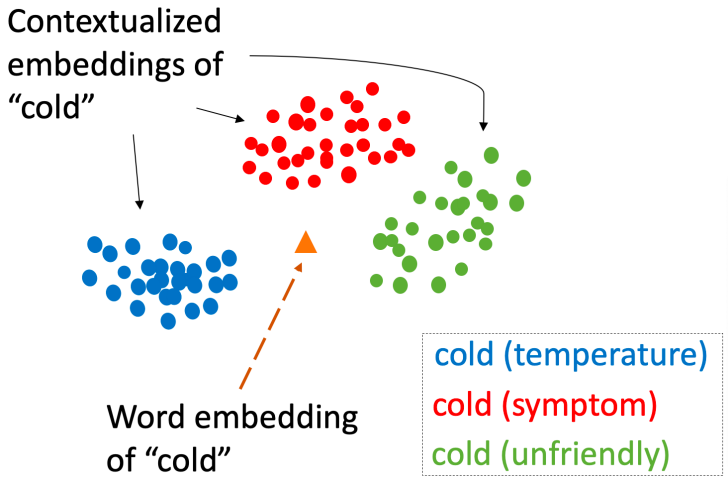


The parameters of RNN are optimized using a downstream task

Contextualized Word Embedding

- Issues of Word2Vec embedding
  - Word2Vec assigns every word with a fixed embedding
  - However, words often exhibit different meanings when placed under different contexts, *e.g.*,
    1) open a *bank* account
    2) on the river *bank*

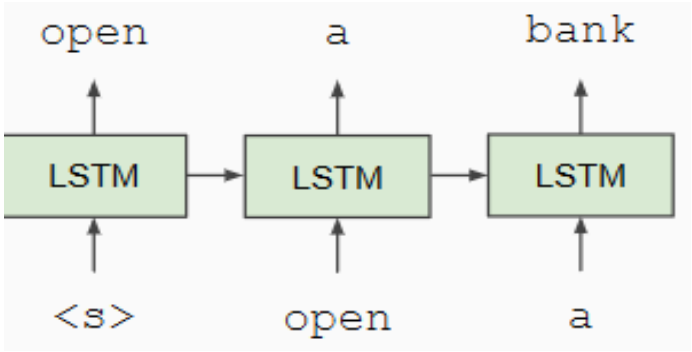
- The embedding of a word should *vary w.r.t. the context* under which it is placed



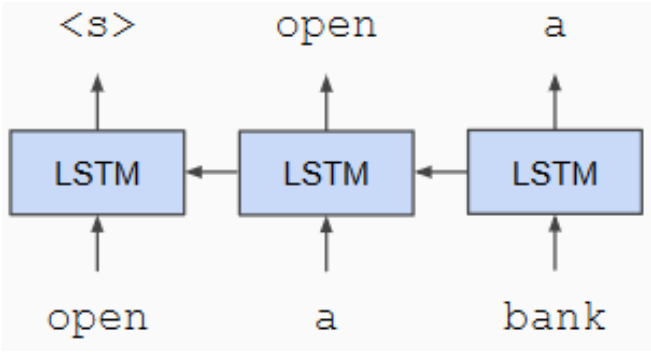
- ELMo

1) First, train a bidirectional RNN on a large corpus

2) Then, pass interested sentences through the pre-trained RNN

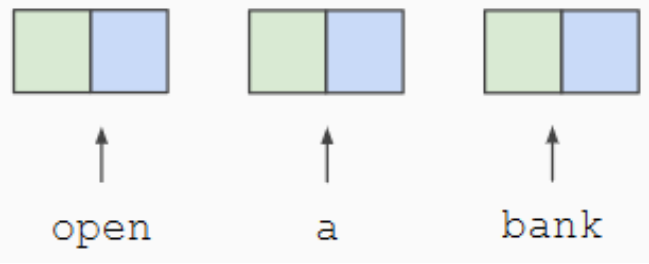


Left-to-right



Right-to-left

3) The final word embeddings are obtained as the concatenation of hidden states of the pre-trained RNN



Outline

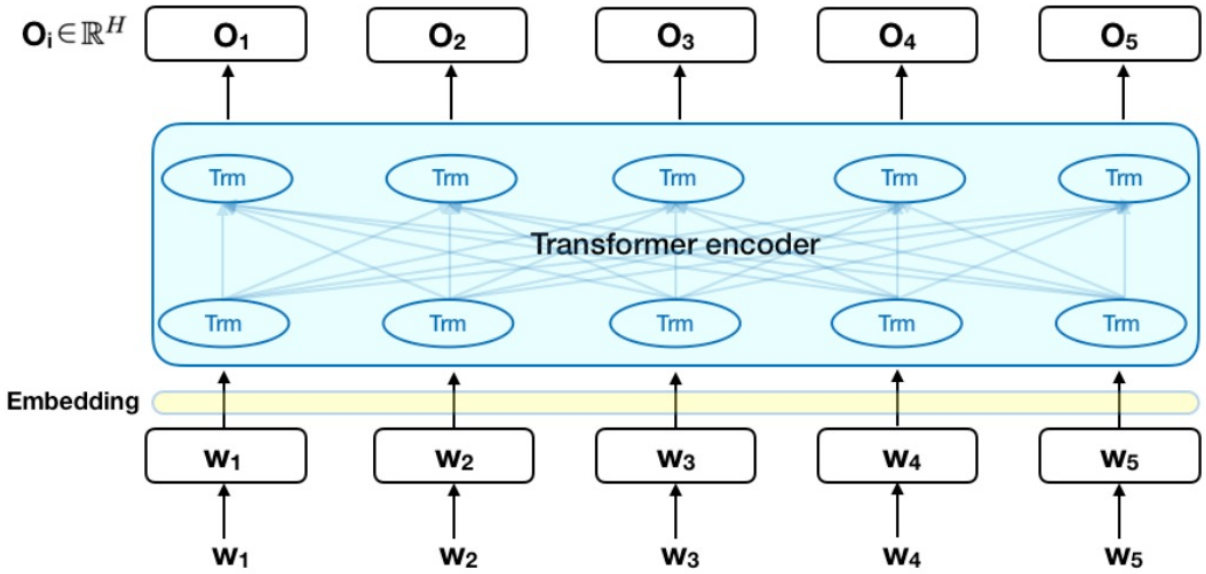
- One-Hot Representation
- Word2Vec Representation
- BERT-Based Representation

- Issues of ELMo embedding

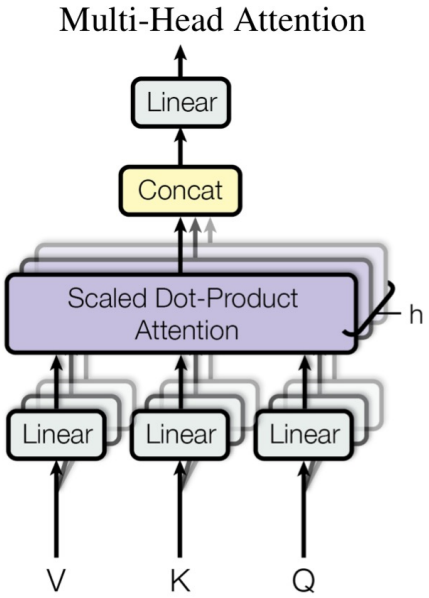
  1) RNN is difficult to capture long-term dependencies of words in documents

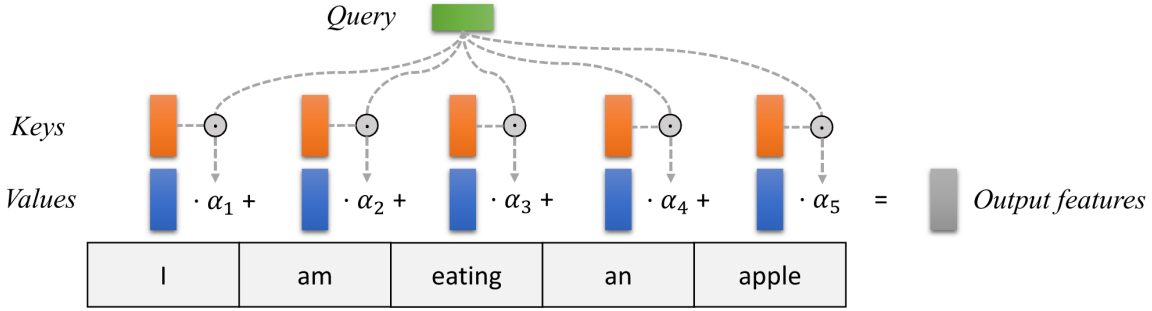
  2) RNN only admits sequential execution, making it difficult to exploit the *parallel computation resources* in GPUs

- Instead of using left-to-right or right-to-left sequential structures, Bidirectional Encoder Representations from Transformers (BERT) adopts a transformer-based 'fully-connected' structure

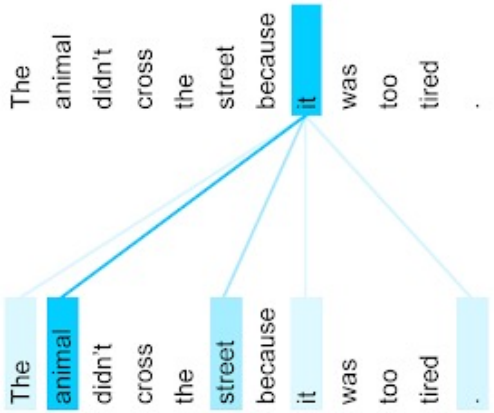


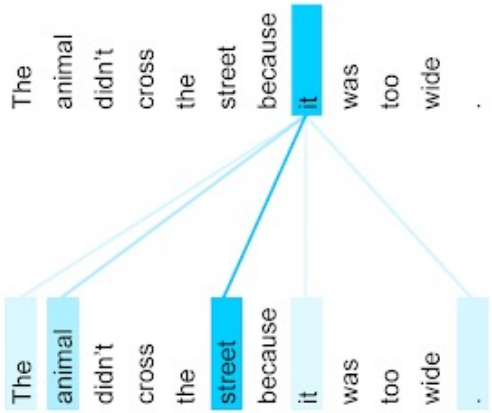
- Different from the fully connection in MLP, the fully-connected structure here is established on a module named ‘transformer’





➢ Example: notice the variation of relevant words of word ‘it’





Pre-Training Tasks

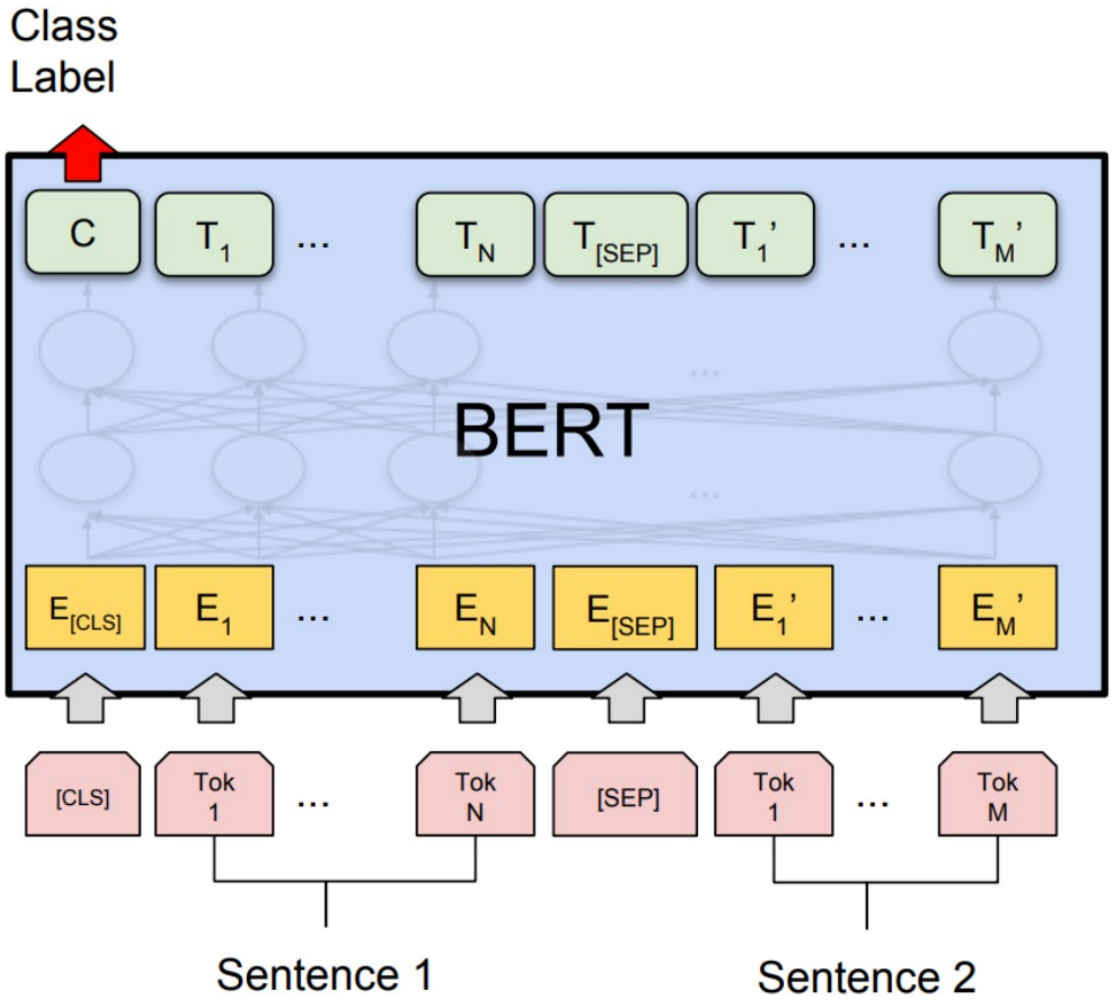
- The model is pre-trained on an extremely large corpus

1) Pre-Training Task 1: Masked Language Model

  - For every input sequence, randomly select 15% of tokens
    - Replace 80% of the selected tokens with the [MASK] token
    - Replace 10% of the selected tokens with a random word
    - Keep the rest 10% of the selected token untouched
  - Objective: training the BERT model to predict the true tokens at the positions of masked tokens

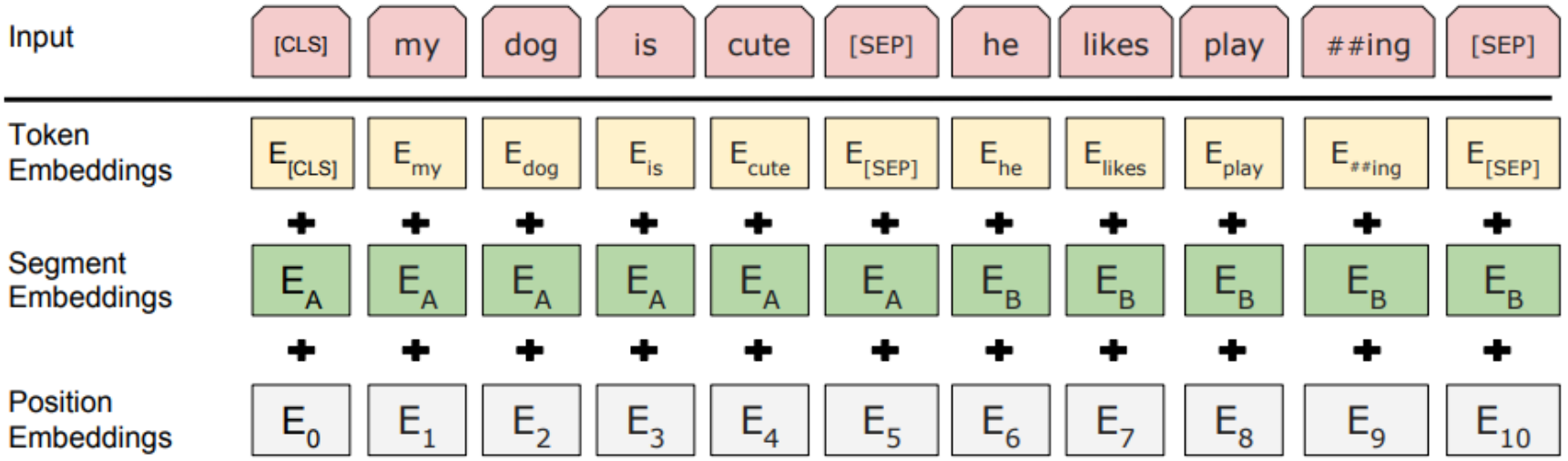
2) Pre-Training Task 2: Next Sentence Prediction

Predict whether sentence B is the next sentence of sentence A


Class
Label
C
T1
...
TN
T[SEP]
T1'
...
TM'
BERT
E[CLS]
E1
...
EN
E[SEP]
E1'
...
EM'
[CLS]
Tok
1
...
Tok
N
[SEP]
Tok
1
...
Tok
M
Sentence 1
Sentence 2

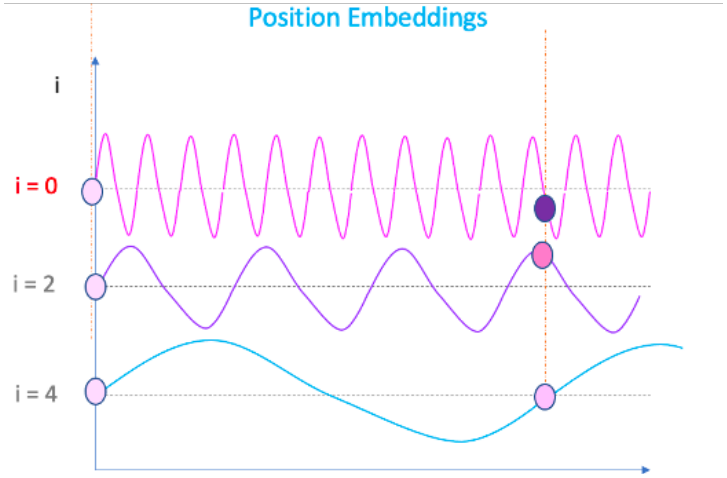
- Input to the BERT model

Sum of three types of embedding



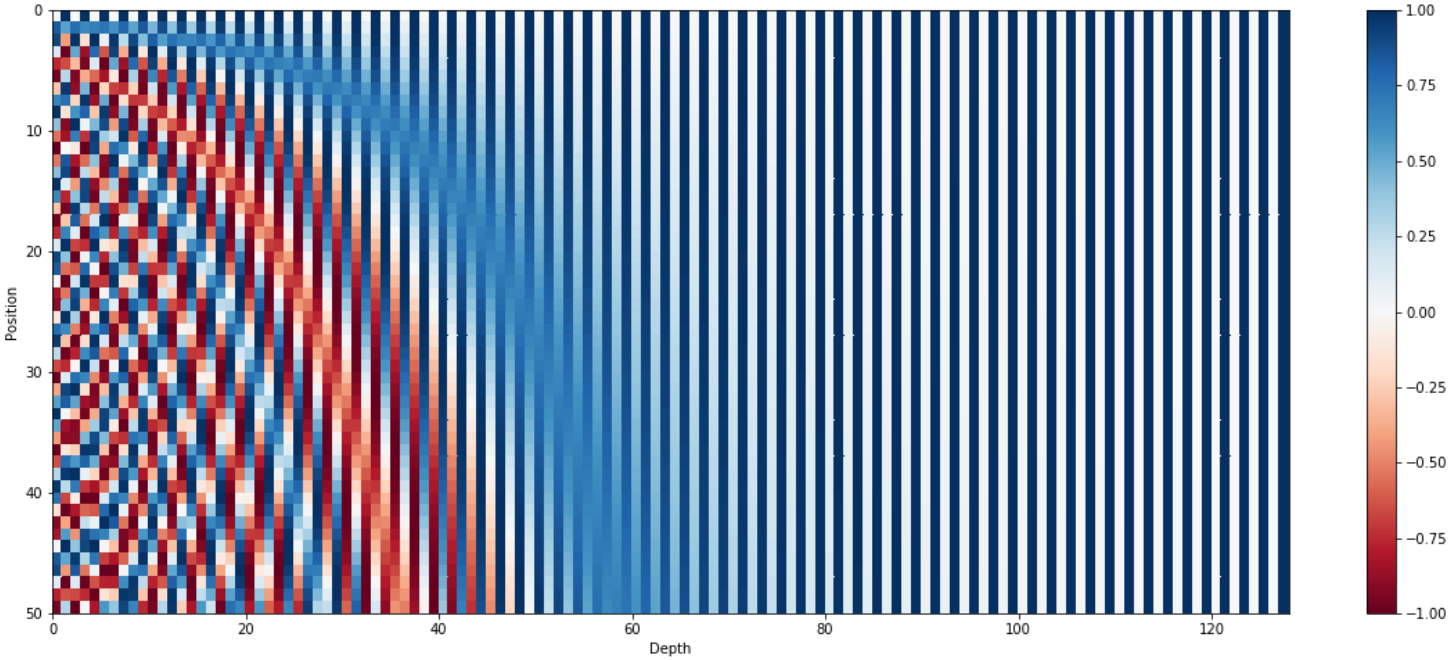
- Position embedding

$$PE(pos, i) = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

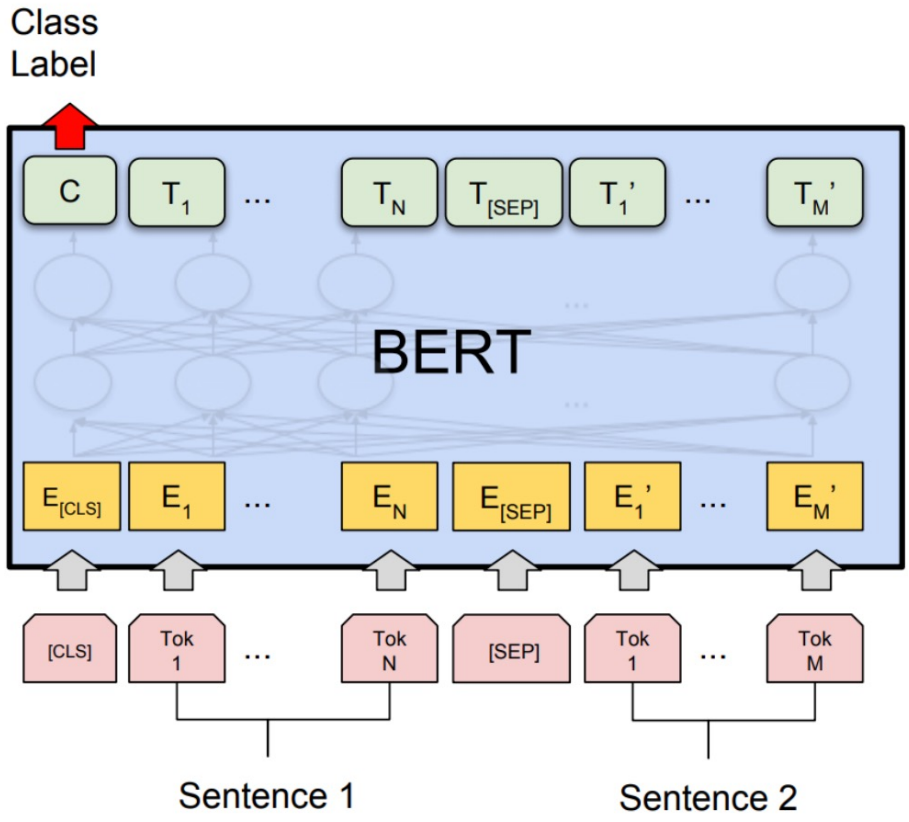


Position of word in a sentence

Word Position



- To obtain word and sentence embedding, pass the interested sentences through the pre-trained BERT model



- Outputs from BERT are the contextualized embeddings of words and sentences