Topics of Data Engineering

Session 11

Masaomi Kimura

Question

- Fill in the blank (what word should be filled with?)

I had a lunch in the park.
I saw a white ____________ barking and running around.

The answer should be “dog”, because …

A dog barks.
A dog runs around.
A dog can be white.

white ___dog____ barking and running around

The meaning of a word is
highly related to words around it

Other vector space approach: word2vec

- Assign a vector to a word
  - The vector should represent the meaning of the word
  - Addition/subtraction of the vectors

$$\boldsymbol{v}(dog) - \boldsymbol{v}(adult) + \boldsymbol{v}(baby) = \boldsymbol{v}(puppy)$$

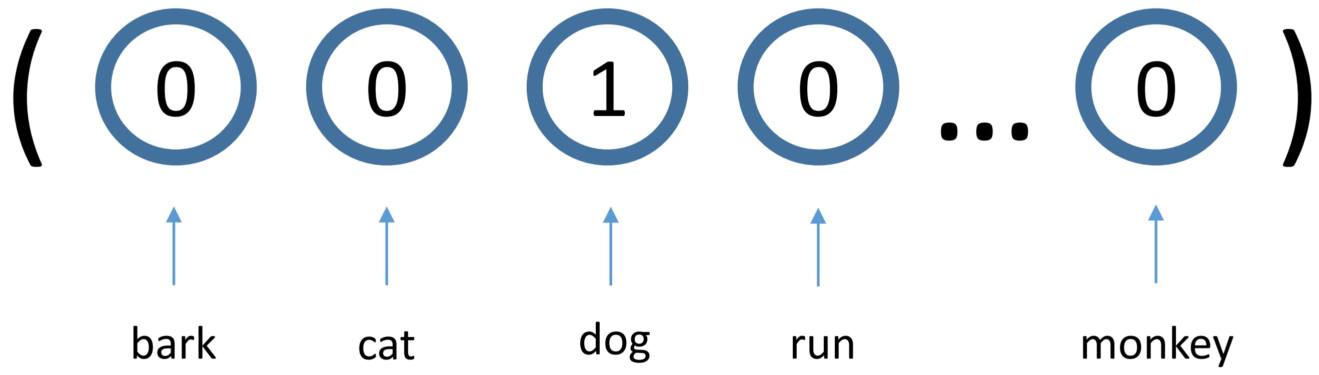
- Similarity of words are proportional to an inner product of their vectors

$$\cos\theta(dog, puppy) = \frac{\boldsymbol{v}(dog)^{\boldsymbol{T}}\boldsymbol{v}(puppy)}{|\boldsymbol{v}(dog)||\boldsymbol{v}(puppy)|}$$

The first step: one-hot vector

- Design a vector whose elements represent the existence of a word and take 1 or 0.

$$\boldsymbol{x}(dog) =$$



Weight matrix W

$$\boldsymbol{W} =$$

$$\begin{pmatrix} \underset{\overset{\uparrow}{\text{bark}}}{\boldsymbol{v}(bark)} & \underset{\overset{\uparrow}{\text{cat}}}{\boldsymbol{v}(cat)} & \underset{\overset{\uparrow}{\text{dog}}}{\boldsymbol{v}(dog)} & \underset{\overset{\uparrow}{\text{run}}}{\boldsymbol{v}(run)} \ldots & \underset{\overset{\uparrow}{\text{monkey}}}{\boldsymbol{v}(monkey)} \end{pmatrix}$$

$$\boldsymbol{v} = W\boldsymbol{x}$$

e.g.) $\boldsymbol{v}(dog) = W\boldsymbol{x}(dog)$

Soft max

- If $a_1, a_2, \cdots a_n$ are given, we can design a "max" function as

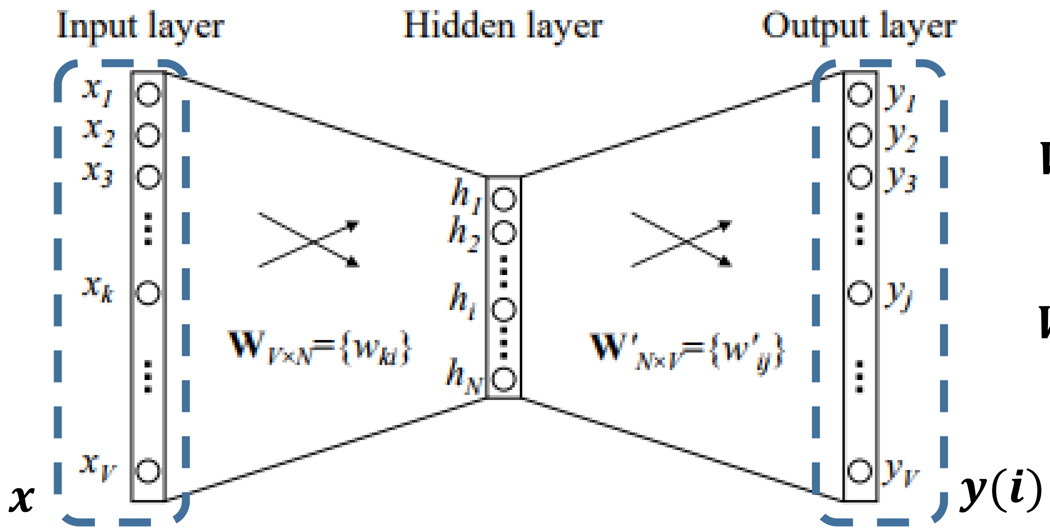
$$\max(a_i) = \begin{cases} 1 & (a_i \text{ is max}) \\ 0 & (\text{otherwise}) \end{cases}$$

- In the context of deep learning, we usually use its differentiable version:

$$\text{softmax}(a_i) = \frac{\exp(a_i)}{\sum_i \exp(a_i)}$$

Classifier of “related words”

$$\mathrm{P}(y(i)|x) = \frac{\exp(y(i)^T \boldsymbol{W'}^T \boldsymbol{W} x)}{\sum_i \exp(y(i)^T \boldsymbol{W'}^T \boldsymbol{W} x)}$$



$\boldsymbol{W}$:one-hot vector for an input word to a word vector

$\boldsymbol{W'}$: a word vector to one-hot vector for a related word

Xin Rong: word2vec Parameter Learning Explained, https://arxiv.org/pdf/1411.2738.pdf

Skip-gram

- If a word is given, we assume that we can guess its related words.

$$P(y(1), y(2), y(3), \cdots, y(C) | x) = \prod_i P(y(i)|x)$$

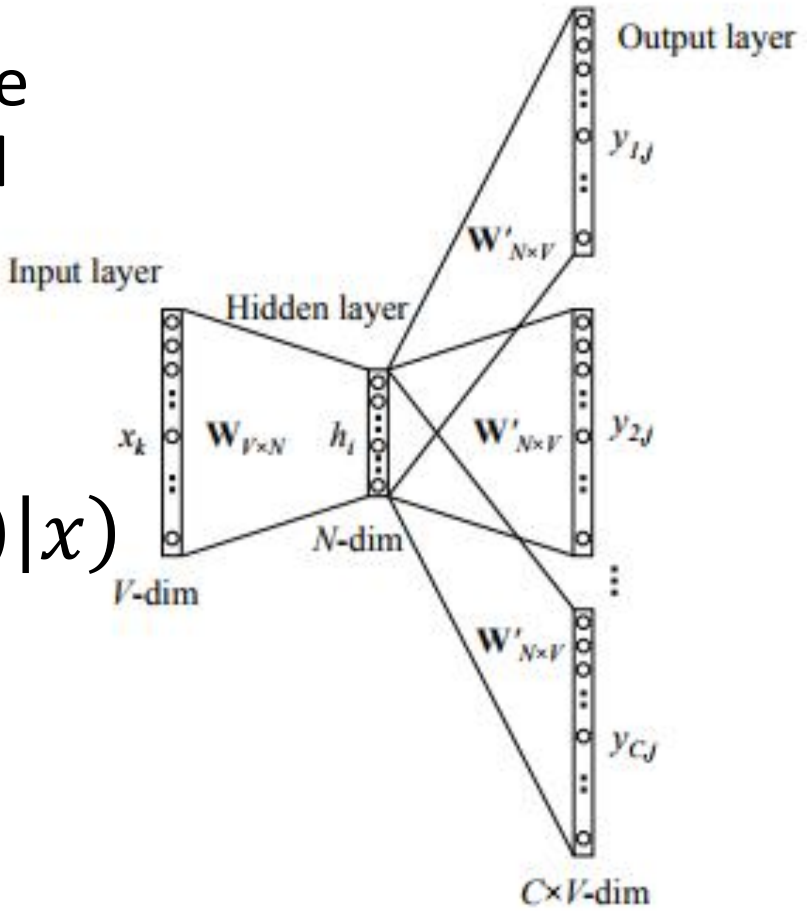


Figure 3: The skip-gram model.

Skip-gram

- If the neighbor words are highly related to a given word, the probability $\mathrm{P}(y(1), y(2), y(3), \cdots | x)$ should be large.

$$\mathrm{P}(y(1), y(2), y(3), \cdots, y(C)|x) = \prod_i \frac{\exp(y(i)^T \boldsymbol{W'}^T \boldsymbol{W} x)}{\sum_j \exp(y(i)^T \boldsymbol{W'}^T \boldsymbol{W} x)}$$

- We need to adjust elements of $\boldsymbol{W}$ and $\boldsymbol{W}'$ to obtain such probabilities.

Loss function

- The function that should be minimum for ideal configuration of the model.
- For skip-gram, the function is:

$$\mathrm{E} = -\log \mathrm{P}(y(1), y(2), y(3), \cdots, y(C) | x)$$

$$= -\log\left(\prod_i \frac{\exp(y(i)^T \boldsymbol{W'}^T \boldsymbol{W} x)}{\sum_j \exp(y(j)^T \boldsymbol{W'}^T \boldsymbol{W} x)}\right)$$

$$= -\sum_i y(i)^T \boldsymbol{W'}^T \boldsymbol{W} x + C \cdot \log\left(\sum_j \exp(y(j)^T \boldsymbol{W'}^T \boldsymbol{W} x)\right)$$

Loss function (cont'd)

- We simplify the notation by use of vectors:

$$\boldsymbol{v}_x = \boldsymbol{W}\boldsymbol{x}$$

$$\boldsymbol{v'}_i = \boldsymbol{W'}\boldsymbol{y}(\boldsymbol{i})$$

$$\mathrm{E} = -\sum_i y(i)^T \boldsymbol{W'}^T \boldsymbol{W} x + C \cdot \log\left(\sum_j \exp\left(y(j)^T \boldsymbol{W'}^T \boldsymbol{W} x\right)\right)$$

$$= -\sum_i \boldsymbol{v'}_i^T \boldsymbol{v}_x + C \cdot \log\left(\sum_j \exp(\boldsymbol{v'}_j^T \boldsymbol{v}_x)\right)$$

Optimization

- Iteration to get optimal v' is the same as the one for neural networks (for i=1,2,…,C):

$$\boldsymbol{v}'^{new}{}_i = \boldsymbol{v}'_i - \epsilon \frac{\partial E}{\partial \boldsymbol{v}'_i}$$

$$= \boldsymbol{v}'_i - \epsilon \left(-\sum_j \frac{\partial \boldsymbol{v}'^{\boldsymbol{T}}_j}{\partial \boldsymbol{v}'_i} \, \boldsymbol{v}_x \; + C \cdot \frac{\exp(\boldsymbol{v}'^{\boldsymbol{T}}_i \boldsymbol{v}_x)}{\sum_j \exp(\boldsymbol{v}'^{\boldsymbol{T}}_j \boldsymbol{v}_x)} \boldsymbol{v}_x \right)$$

$$= \boldsymbol{v}'_i - \epsilon \left(-1 + C \cdot \frac{\exp(\boldsymbol{v}'^{\boldsymbol{T}}_i \boldsymbol{v}_x)}{\sum_j \exp(\boldsymbol{v}'^{\boldsymbol{T}}_j \boldsymbol{v}_x)} \right) \boldsymbol{v}_x$$

$$= \boldsymbol{v}'_i - \epsilon \mathrm{EI}_i \; \boldsymbol{v}_x$$

Essentially…

- If we rewrite the probability $\mathrm{P}(y(1), y(2), y(3), \cdots, y(C)|x)$ as:

$$\mathrm{P}(y(1), y(2), y(3), \cdots, y(C)|x) = \frac{\exp(\sum_i \boldsymbol{v'}_i^T \boldsymbol{v}_x)}{Z^n}$$

- Since the probability should be large for frequently co-occurring words,

$$\boldsymbol{v}_x \approx \sum_f \boldsymbol{v'}_f^T$$

f: frequently co-occurring words

Distributed representation: Examples

If ‘dog’ frequently co-occurs with ‘bark’ and ‘run’ then

$$\boldsymbol{v}(dog) \approx \boldsymbol{v}'(bark) + \boldsymbol{v}'(run)$$

If there are relationships:

$$\boldsymbol{v}(adult) \approx \boldsymbol{v}'(bark)$$

$$\boldsymbol{v}(baby) \approx \boldsymbol{v}'(yap)$$

$$\boldsymbol{v}(puppy) \approx \boldsymbol{v}'(yap) + \boldsymbol{v}'(run)$$

then

$$\boldsymbol{v}(dog) - \boldsymbol{v}(adult) + \boldsymbol{v}(baby) = \boldsymbol{v}(puppy)$$

We can expect …

- Sum of the vectors in a sentence expresses its meaning

I saw a white dog barking and running around.

$$\boldsymbol{v}(I)+\boldsymbol{v}(see)+\boldsymbol{v}(white)+\boldsymbol{v}(dog)$$
$$+\boldsymbol{v}(bark)+\boldsymbol{v}(run)+\boldsymbol{v}(around)$$

LSTM

$$\boldsymbol{f} = \sigma\left(W_h^f \boldsymbol{h}^{(t-1)} + W_x^f \boldsymbol{x} - \theta^f\right)$$
$$\boldsymbol{i} = \sigma\left(W_h^i \boldsymbol{h}^{(t-1)} + W_x^i \boldsymbol{x} - \theta^i\right)$$
$$\boldsymbol{o} = \sigma\left(W_h^o \boldsymbol{h}^{(t-1)} + W_x^o \boldsymbol{x} - \theta^o\right)$$
$$\widetilde{\boldsymbol{C}} = \sigma\left(W_h^C \boldsymbol{h}^{(t-1)} + W_x^C \boldsymbol{x} - \theta^C\right)$$

$$\boldsymbol{C}^{(t)} = \boldsymbol{f} \odot \boldsymbol{C}^{(t-1)} + \boldsymbol{i} \odot \widetilde{\boldsymbol{C}}$$
$$\boldsymbol{h}^{(t)} = \boldsymbol{o} \odot \tan\left(C^{(t)}\right)$$

Note : The symbol ⊙ denotes Hadamard product.

Hadamard product

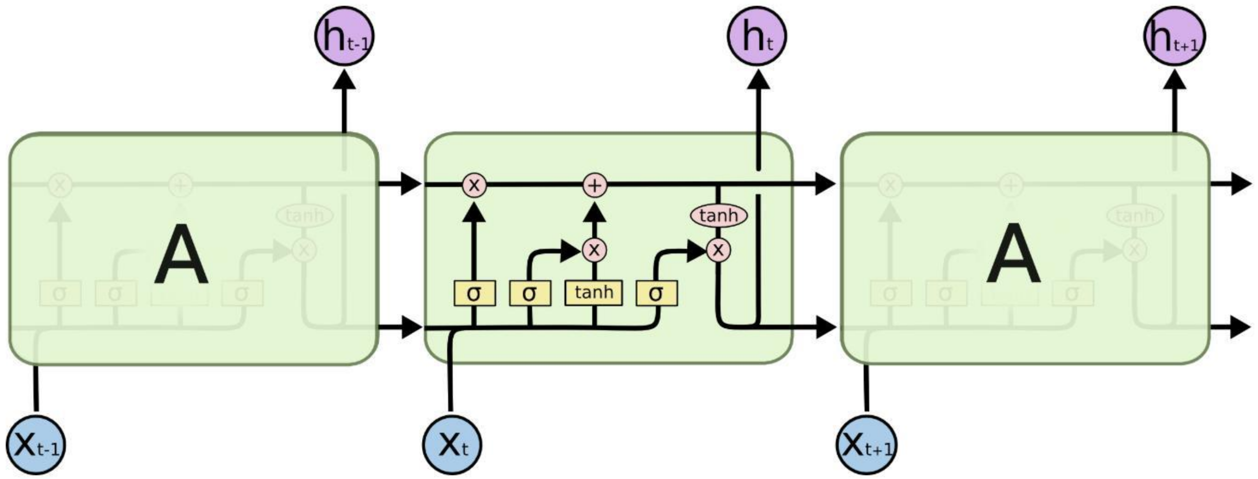
- We have two vectors: $\boldsymbol{v}_1 = \begin{pmatrix} a_1 \\ b_1 \end{pmatrix}$ and $\boldsymbol{v}_2 = \begin{pmatrix} a_2 \\ b_2 \end{pmatrix}$.
- Hadamard product is a element wise product:

$$\boldsymbol{v}_1 \odot \boldsymbol{v}_2 = \begin{pmatrix} a_1 a_2 \\ b_1 b_2 \end{pmatrix}$$

- An example

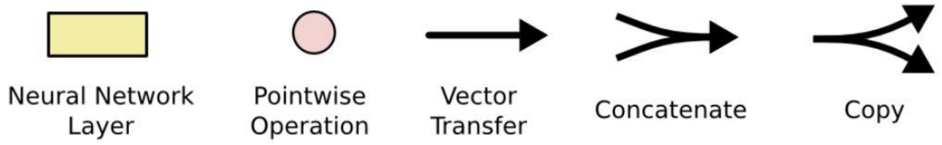
$$\begin{pmatrix} 2 \\ 3 \end{pmatrix} \odot \begin{pmatrix} 5 \\ 3 \end{pmatrix} = \begin{pmatrix} 10 \\ 9 \end{pmatrix}$$

Sequential input to LSTM



The repeating module in an LSTM contains four interacting layers.

Don't worry about the details of what's going on. We'll walk through the LSTM diagram step by step later. For now, let's just try to get comfortable with the notation we'll be using.



The essential part of LSTM

$$\boldsymbol{C}_t = \boldsymbol{f} * \boldsymbol{C}_{t-1} + \boldsymbol{i} * \widetilde{\boldsymbol{C}}_t$$

Weight to forget the previous vector

Weight for input

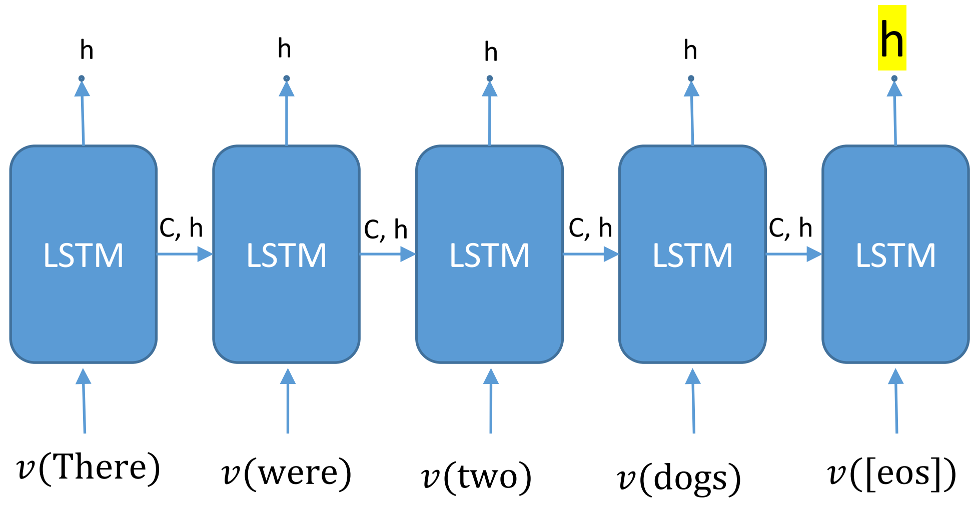
The structure is similar to the sum of word vectors in a sentence:

$$\boldsymbol{v}(\text{"}I\ see\ dog\text{"}) = \boldsymbol{v}(\text{"}I\ see\text{"}) + \boldsymbol{v}(dog)$$

⇨ Combination of distributed expressions and LSTM is useful for NLP (and text mining!)

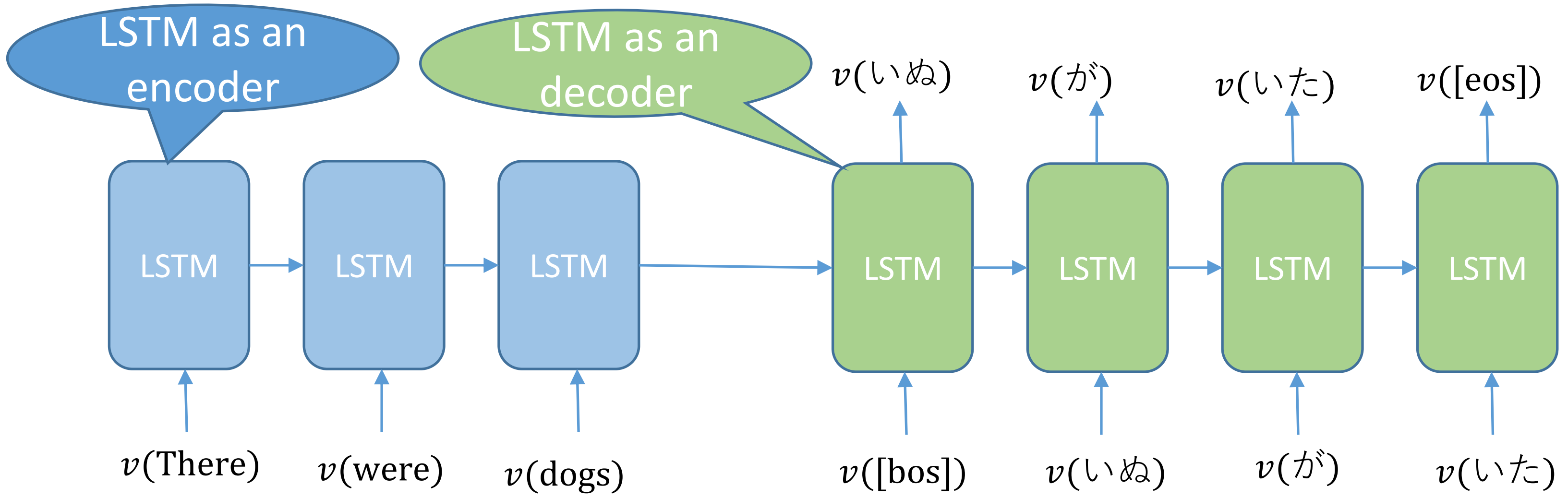
Sequential input to LSTM

- Word embedding vectors are sequentially input to the LSTM model
- The output vector for the sequential data is obtained after *eos* (end of sentence) is input



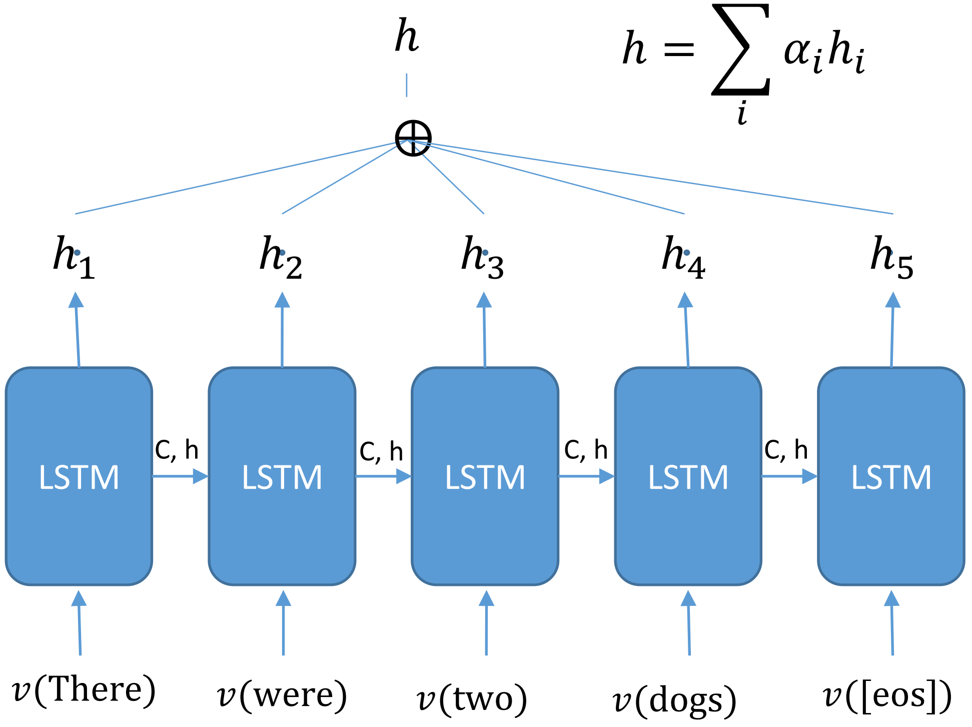
Application: translation

- Two LSTM models are used for encoder and decoder
  - It does not work well for a long sentence, because LSTM tends to forget input several steps ago



Attention mechanism

- The output of LSTM for each step is weighted and summed to get an output that covers the whole of steps


h
$h = \sum_i \alpha_i h_i$
h_1
h_2
h_3
h_4
h_5
LSTM
C, h
v(There)
v(were)
v(two)
v(dogs)
v([eos])