CS 6180

Review from last time

LSTMs (specific type of RNNs)

Machine Translation

task of generating a sentence y from one language to another.

French Il faut profiter du moment présent

English You have to enjoy the present moment

(Carpe Diem) (Latin)

Learn

model

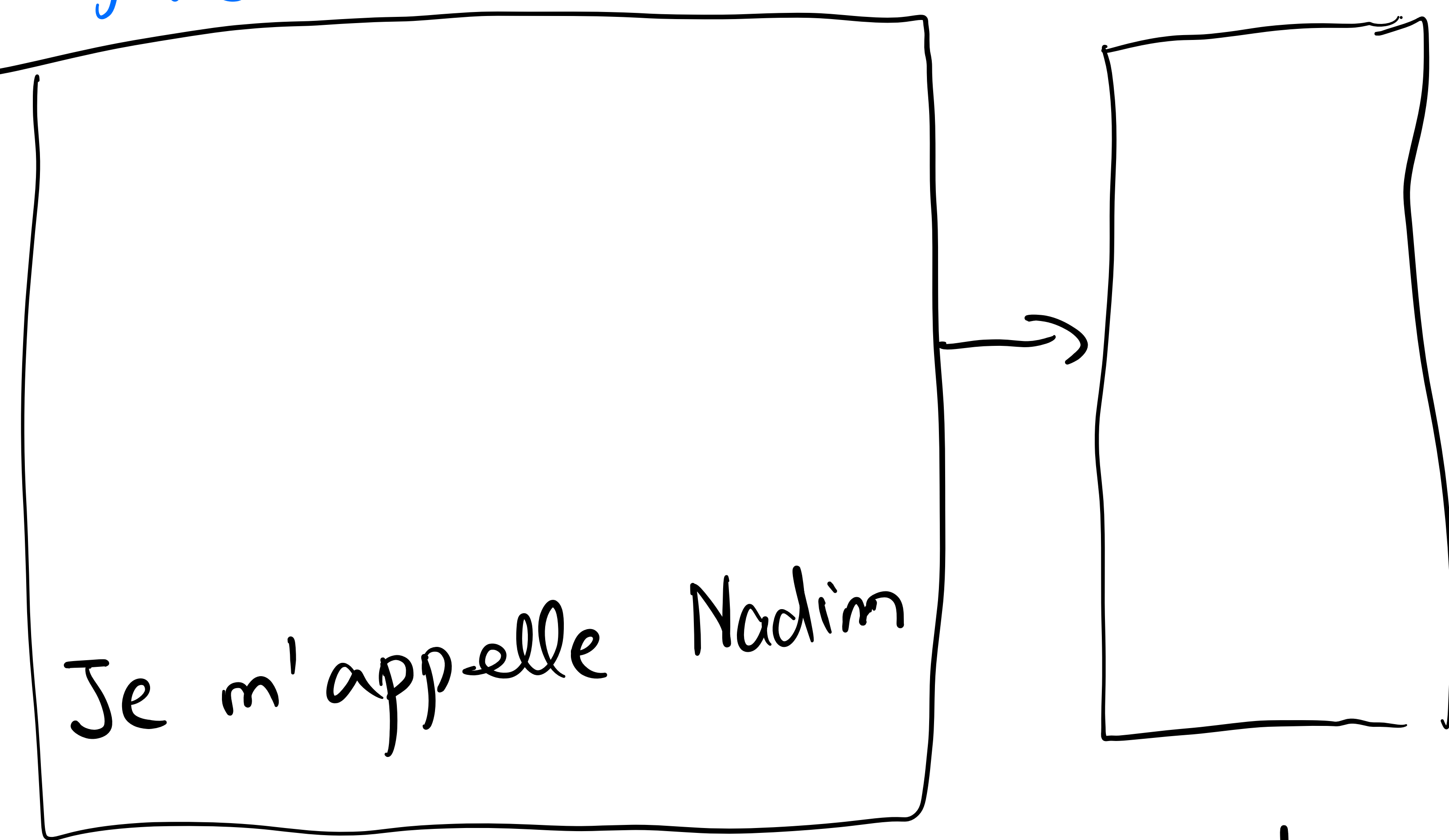
$$p(y \mid x)$$

(sentence in French

in English

Goal: maximize this probability

Neural Machine Translation



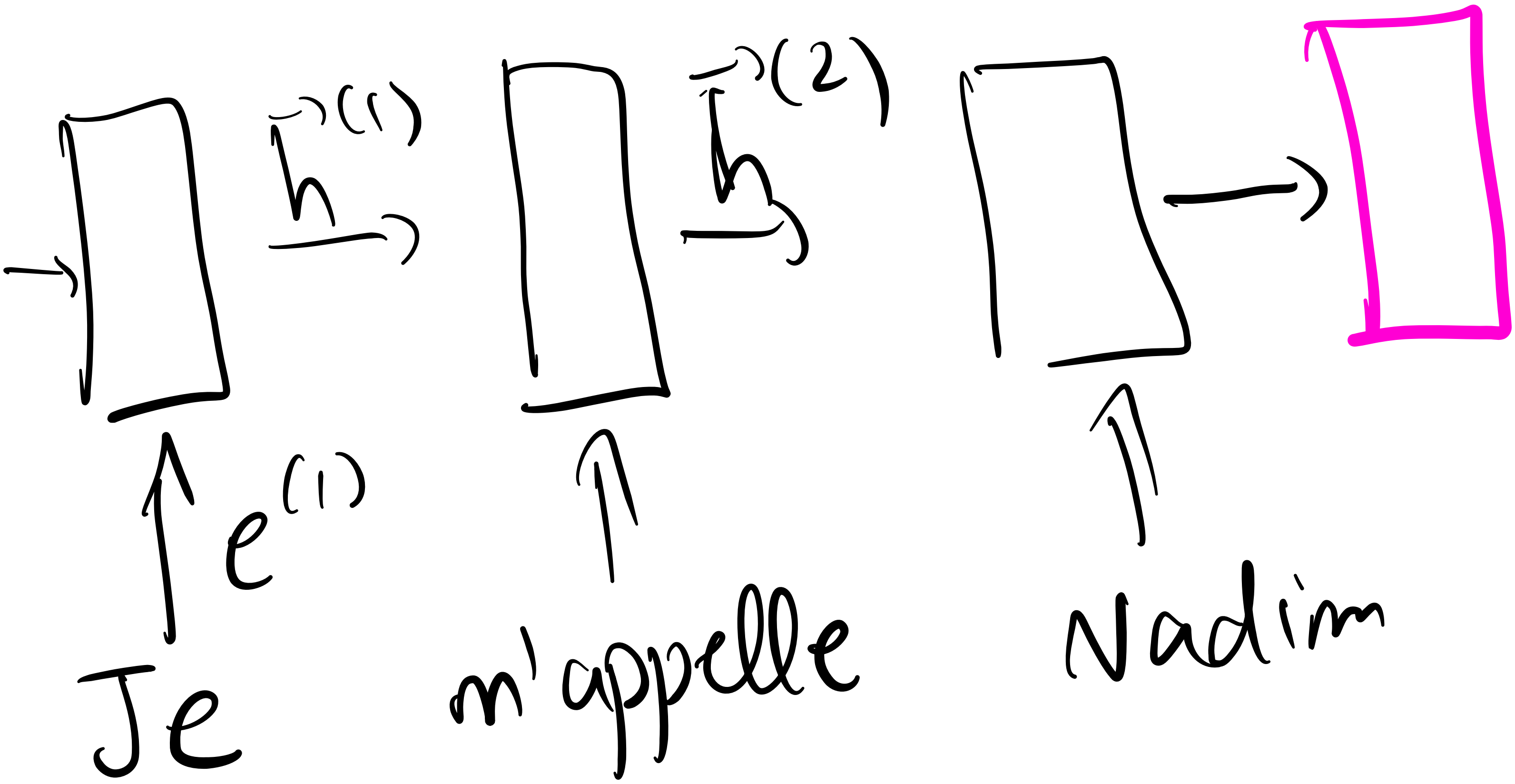
Encoder

Decoder

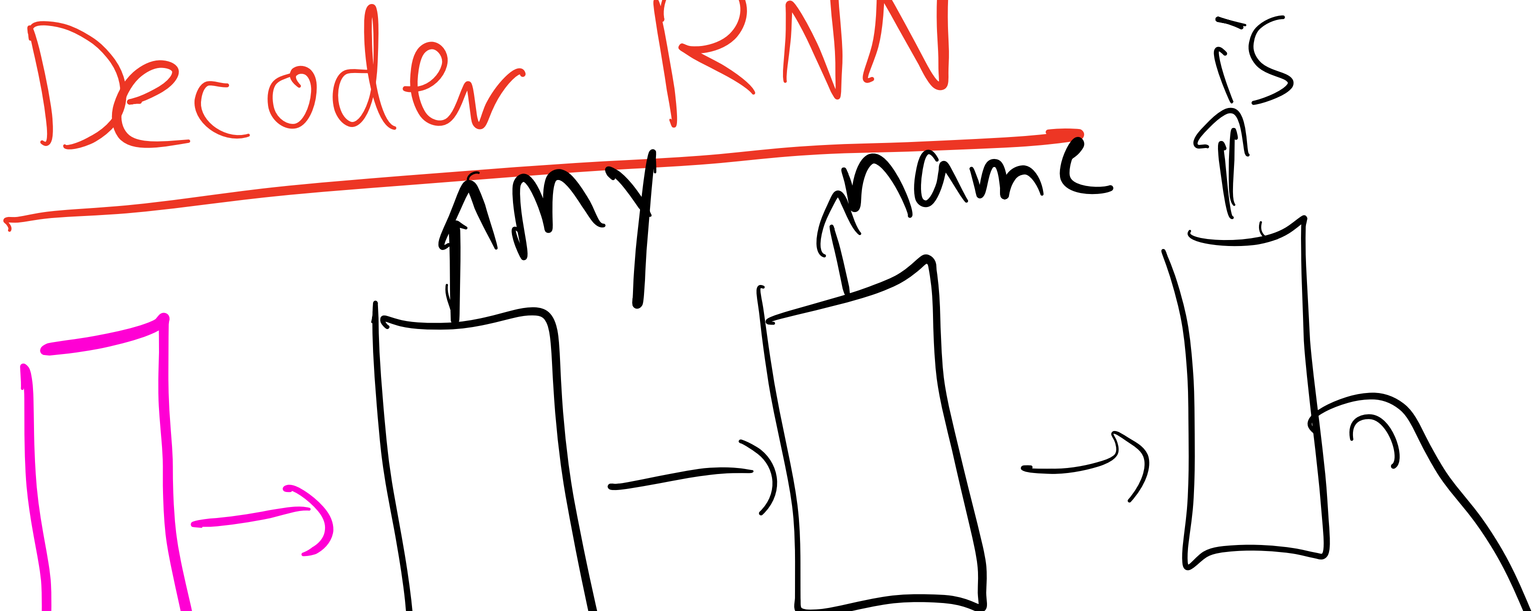
responsible of the translation

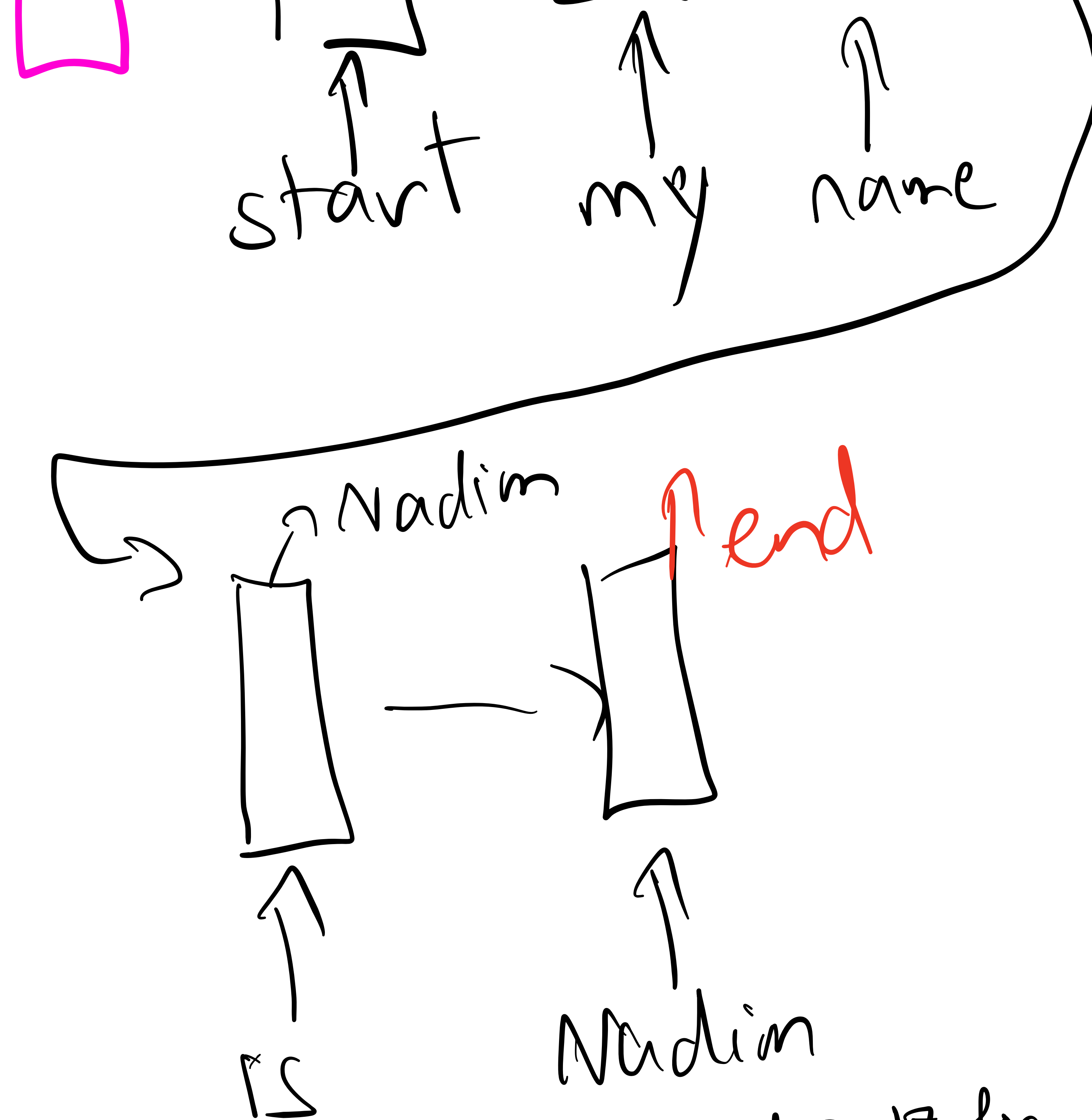
Encoder Component

(Encoder RNN)



Decoder RNN





* they were the best around 2017 for translation
* not parallelizable
  computations can be expensive

* O(sequence length) info stored into one vector

$\Rightarrow$ **Attention** can help

(image on board)

* dot product between words in the source sentence and word to be generated

$\vec{q_i}$

$\vec{e_1}$ $\vec{e_2}$ $\vec{e_3}$

wǒ jiào Nàdìmǔ

$$\left.\begin{array}{l} \vec{q_i}^T \vec{e_1} \\ \vec{q_i}^T \vec{e_2} \\ \vec{q_i}^T \vec{e_3} \end{array}\right\} \text{attention scores}$$

apply softmax to get weights $\longrightarrow$

$$\alpha_1 = \frac{\exp(\vec{q_i}^T \vec{e_1})}{\sum_j \exp(\vec{q_i}^T \vec{e_j})}$$

$$\alpha_2 = \frac{\exp(\vec{q_i}^T \vec{e_2})}{\sum_j \exp(\vec{q_i}^T \vec{e_j})}$$

$$\alpha_3 = \frac{\exp(\vec{q_i}^T \vec{e_3})}{\sum_j \exp(\vec{q_i}^T \vec{e_j})}$$

$$\text{Output} = \alpha_1 \vec{e}_1 + \alpha_2 \vec{e}_2 + \alpha_3 \vec{e}_3$$

we are asking ~~too~~ much from the word embeddings

meaning of the word

similarity to other words

similarity in other languages

$$\vec{e}_i \to \text{key}, \quad \vec{e}_i \to \text{query}, \quad \vec{e}_i \to \text{value}$$

$$\vec{k}_i = \boxed{K}\vec{e}_i \qquad \vec{q}_i = \boxed{Q}\vec{e}_i \qquad \vec{v}_i = \boxed{V}\vec{e}_i$$

$d \times d$ $\qquad$ $d \times d$ $\qquad$ $d \times d$

matrices that we have to learn (parameters)

dot products :

$$\vec{q}_i^{\,T} \vec{k}_j$$

$$= (Q\vec{e}_i)^T (K\vec{e}_j)$$

$$= \vec{e}_i^{\,T} \underbrace{\boxed{Q^T K}}_{M} \vec{e}_j$$

to train the model computationally, separating the Q and K leads to a much more efficient implementation.. (Let's see why)

$$\underset{d_Q \times 1}{\vec{q}_i} = \underset{\boxed{d_Q \times d}}{\boxed{Q}}\vec{e}_i \qquad \overset{d_Q \times 1}{\vec{k}_i} = \underset{\boxed{d_Q \times d}}{\boxed{K}}\vec{e}_i$$

what if $d_Q \ll d$?

$\boxed{2 d d_Q}$ total # of parameters for both Q and K

vs

$$M = \underbrace{\underset{d \times d_Q}{Q^T} \; \underset{d_Q \times d}{K}}_{d \times d}$$

d^2 parameters

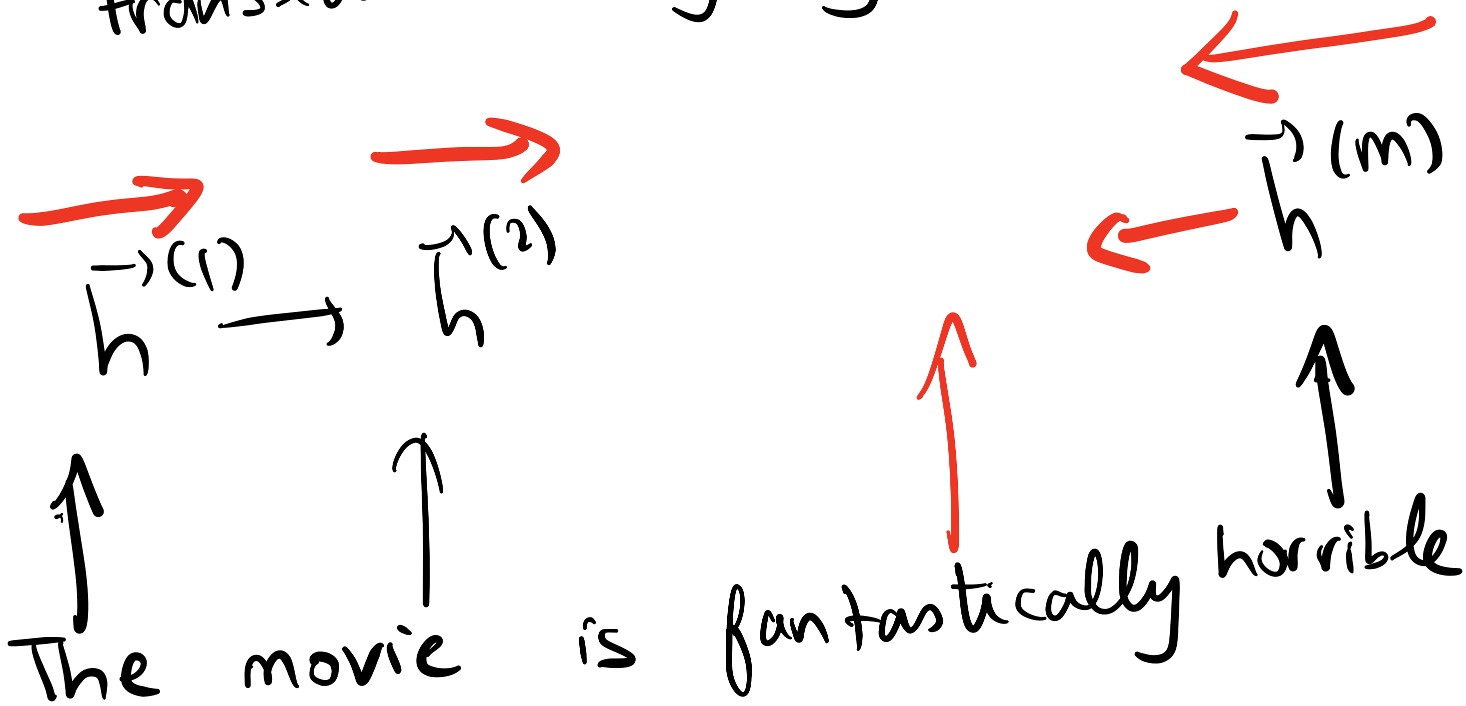
$2 d d_Q$ vs d^2 if $d_Q \ll d$

less parameters to keep track of
more efficient computationally.

(note to Nadim: add two other screenshots)

Attention can be used for Language models in general (not just translation)

translation (using only decoder)



LSTM ($\rightarrow$) left to right

LSTM ($\leftarrow$) right to left

$\overrightarrow{\text{RNN}}$

$$\vec{h}^{(t)} = \sigma\left(\vec{W}_h \vec{h}^{(t-1)} + \vec{W}_e \vec{e}^{(t)} + \vec{b}_1\right)$$

$\overleftarrow{RNN}$

$$\overleftarrow{h}^{(t)} = \sigma\left(W_h \overleftarrow{h}^{(t+1)} + W_e e^{(t)} + b_1\right)$$

$$\text{word } t = \begin{bmatrix} \overrightarrow{h}^{(t)} \\ \overleftarrow{h}^{(t)} \end{bmatrix}$$

Bi-directional LSTMs can apply for encoders

Not for the decoders

NTA for Language model