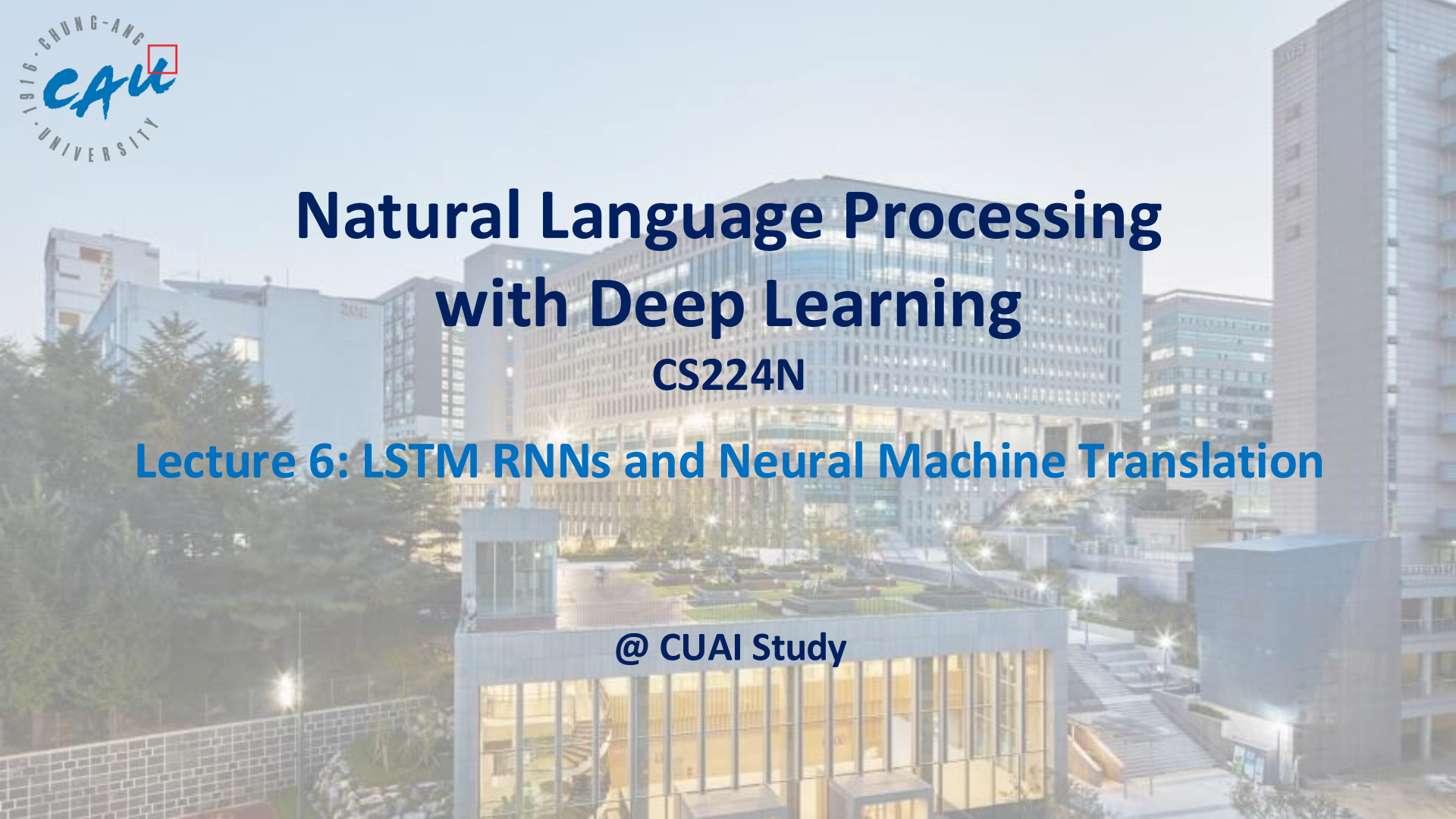


Natural Language Processing with Deep Learning

CS224N

Lecture 6: LSTM RNNs and Neural Machine Translation

@ CUAI Study

Evaluating Language Models

❖ Standard Evaluation Metric: Perplexity

- The lower the perplexity, the better the performance

$$\text{perplexity} = \prod_{t=1}^{T} \left(\frac{1}{P_{\text{LM}}(\boldsymbol{x}^{(t+1)} | \boldsymbol{x}^{(t)}, \ldots, \boldsymbol{x}^{(1)})} \right)^{1/T}$$

Normalized by number of words

Inverse probability of corpus, according to Language Model

n-gram model

Increasingly complex RNNs

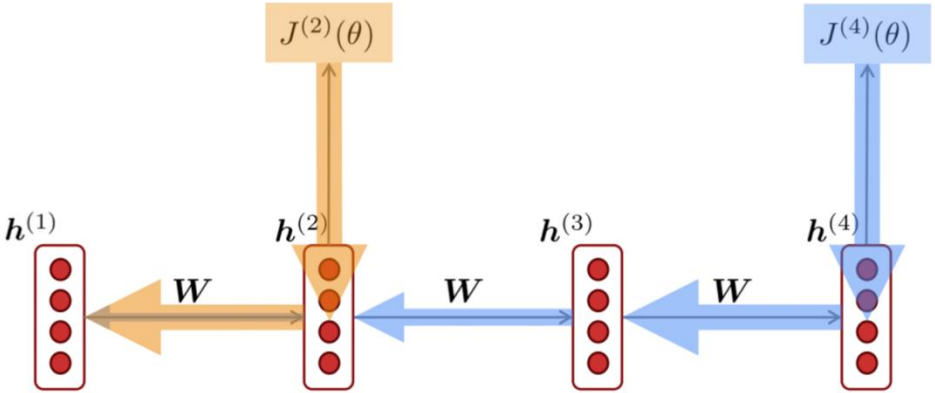
Model	Perplexity
Interpolated Kneser-Ney 5-gram (Chelba et al., 2013)	67.6
RNN-1024 + MaxEnt 9-gram (Chelba et al., 2013)	51.3
RNN-2048 + BlackOut sampling (Ji et al., 2015)	68.3
Sparse Non-negative Matrix factorization (Shazeer et al., 2015)	52.9
LSTM-2048 (Jozefowicz et al., 2016)	43.7
2-layer LSTM-8192 (Jozefowicz et al., 2016)	30
Ours small (LSTM-2048)	43.9
Ours large (2-layer LSTM-2048)	39.8

Perplexity improves (lower is better)

Problems with RNNs

❖ (1) Vanishing Gradient Problem:

- **Gradient signal from far** away is lost because it is much smaller than **gradient signal from close-by**
- So, model weights are basically updated only with respect to **near effects**, not **long-term effects**



Effect of vanishing gradient on RNN-LM

❖ Language Model

- Need to be able to transmit signals a long distance
- Need to model the dependency between words

LM task: *When she tried to print her tickets, she found that the printer was out of toner. She went to the stationery store to buy more toner. It was very overpriced. After installing the toner into the printer, she finally printed her* _____

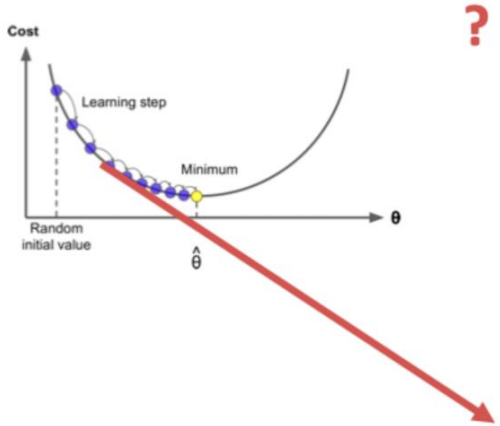
- Unable to predict similar long-distance dependencies at test time
- Can't learn the dependency if the gradient is small

Problems with RNNs

❖ (2) Exploding Gradient Problem

- If the gradient becomes too big, then the SGD update step becomes too big
- Too large a step: a weird and bad parameter configuration >> bad update
- Worst case: Inf or NaN

$$\theta^{new} = \theta^{old} - \underbrace{\overbrace{\alpha}^{\text{learning rate}} \nabla_\theta J(\theta)}_{\text{gradient}}$$



Solution for these gradient problems

❖ Exploding Gradients: Gradient clipping

- If the norm of the gradient is greater than some threshold, scale it down before applying SGD update
- Intuition: Take a step in the same direction, but a smaller step

Algorithm 1 Pseudo-code for norm clipping

$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ **then**
  $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|}\hat{\mathbf{g}}$
end if

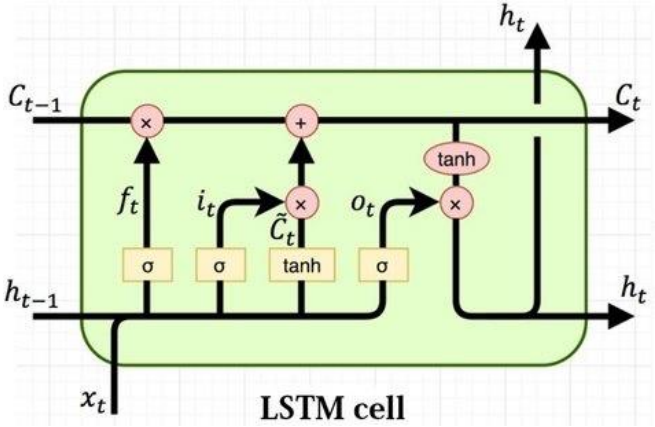
❖ How to fix the vanishing gradient problem?

- It is too difficult for the RNN to learn to preserve information over many timesteps
- ➢ How about designing RNN with **separate memory** which is added to?

Long Short-Term Memory RNNs

❖ LSTMs

- Can **read, erase, and write information** from the cell
- Selection of which information is erased/written/read is controlled by three corresponding **gates**
- Gates are dynamic: their value is computed based on the current context



LSTM cell

$$
\begin{aligned}
i_t &= \sigma\left(x_t U^i + h_{t-1} W^i\right) \\
f_t &= \sigma\left(x_t U^f + h_{t-1} W^f\right) \\
o_t &= \sigma\left(x_t U^o + h_{t-1} W^o\right) \\
\tilde{C}_t &= \tanh\left(x_t U^g + h_{t-1} W^g\right) \\
C_t &= \sigma\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right) \\
h_t &= \tanh(C_t) * o_t
\end{aligned}
$$

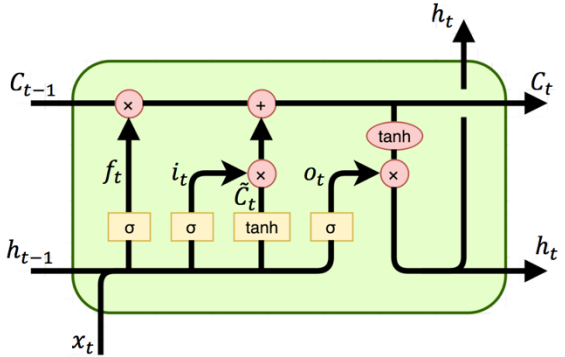
Long Short-Term Memory (LSTM)

❖ What Each Gate Means

- **Forget gate** $f^{(t)}$: Control what is kept vs forgotten, from previous cell state
- **Input gate** $i^{(t)}$: Control what parts of the new cell content are written to cell
- **Output gate** $o^{(t)}$: Control what parts of cell are output to hidden state
- **New cell content** $\hat{c}^{(t)}$: This is the new content to be written to the cell
- **Cell state** $c^{(t)}$: Erase ("forget") some content from the last cell state,

  and write ("input") some new cell content
- **Hidden state** $h^{(t)}$: Read ("output") some content from the cell
- **Sigmoid function** : All gate values are between 0 and 1

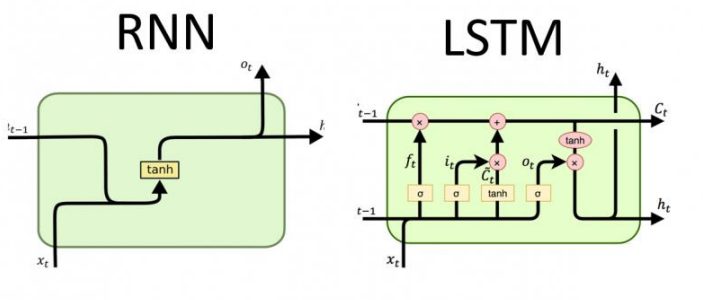
$$
\begin{aligned}
i_t &= \sigma\left(x_t U^i + h_{t-1} W^i\right) \\
f_t &= \sigma\left(x_t U^f + h_{t-1} W^f\right) \\
o_t &= \sigma\left(x_t U^o + h_{t-1} W^o\right) \\
\tilde{C}_t &= \tanh\left(x_t U^g + h_{t-1} W^g\right) \\
C_t &= \sigma\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right) \\
h_t &= \tanh(C_t) * o_t
\end{aligned}
$$



Long Short-Term Memory (LSTM)

❖ How does LSTM solve vanishing gradients?

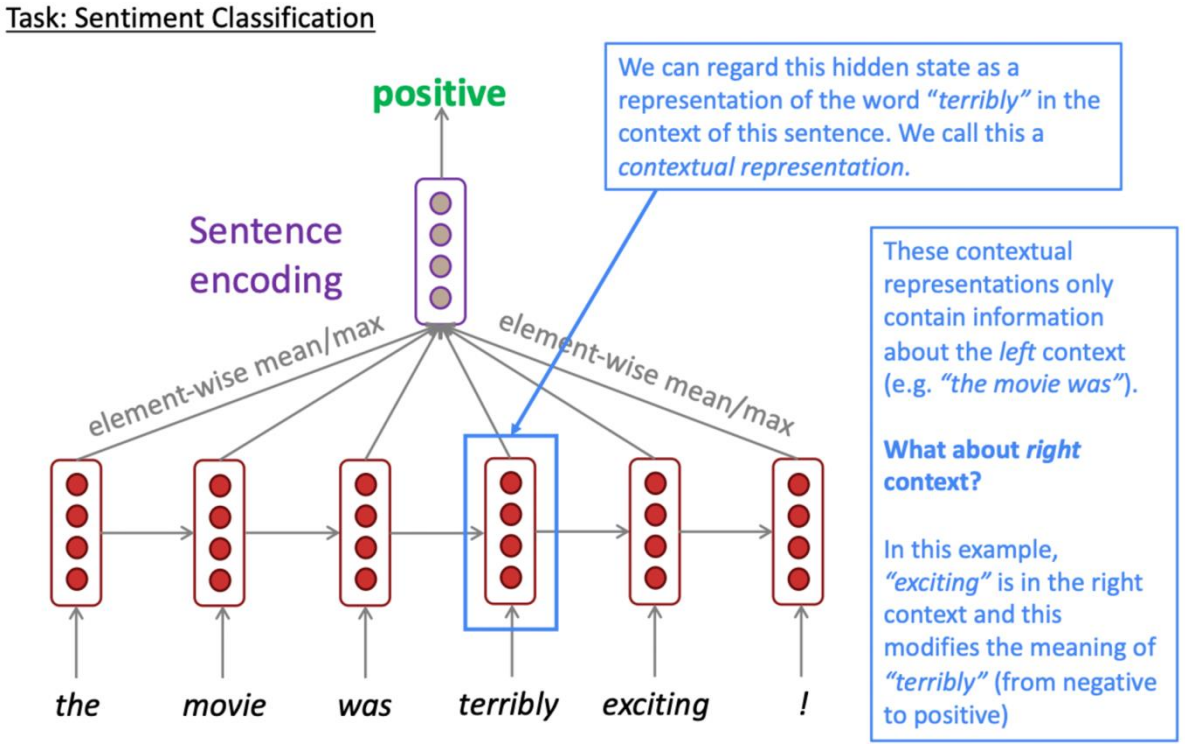
- LSTM architecture makes it much easier for an RNN to preserve information over many timesteps
- It will learn that certain kind of information is useful >> keep carrying forward for at least a while
- Not multiplicative stuff – just add some new information to previous cell



- But **Not** a vanishing/exploding gradient is an RNN problem
- For all neural architectures, especially very deep ones
- RNNs are particularly unstable due to the repeated multiplication by the same weight matrix

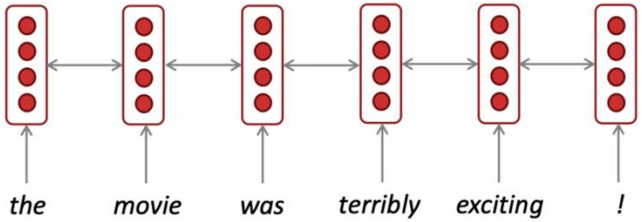
Bidirectional and Multi-Layer RNNs

❖ Motivation



Bidirectional RNNs

❖ Simplified Diagram



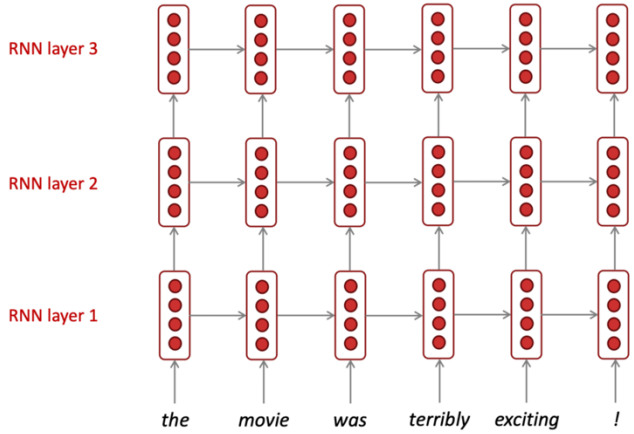
The two-way arrows indicate bidirectionality and the depicted hidden states are assumed to be the concatenated forwards+backwards states

- Note:
  - Only applicable if the **entire input** has been accessible
  - Not applicable to Language Modeling, because in LM – only have left context available

Multi-layer RNNs

❖ Stacked RNNs

- RNNs are already “deep” on one dimension
- Make “deep” in another dimension by **applying multiple RNNs**
- Allow the network to compute more complex representations
  - Work better than just have one layer of high-dimensional encodings
- High-performing RNNs are usually multi-layer



Machine Translation

❖ What is Machine Translation?

- Was actually where NLP started
- **The task of translating a sentence x from one language to a sentence y in another language**
- Two types: Statistical Machine Translation, and Neural Machine Traslation

(source language) *x:* *L'homme est né libre, et partout il est dans les fers*

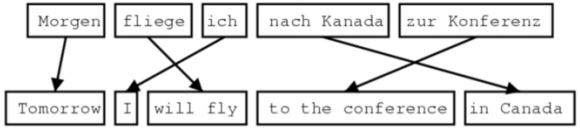
(target language) *y:* *Man is born free, but everywhere he is in chains*

– Rousseau

Machine Translation - 1

❖ Statistical Machine Translation

- Core idea: Learn a **probabilistic model** from **data**



$$\text{argmax}_y P(y|x) = \text{argmax}_y P(x|y)P(y)$$

Translation Model

Models how words and phrases should be translated (*fidelity*). Learned from parallel data.

Language Model

Models how to write good English (*fluency*). Learned from monolingual data.

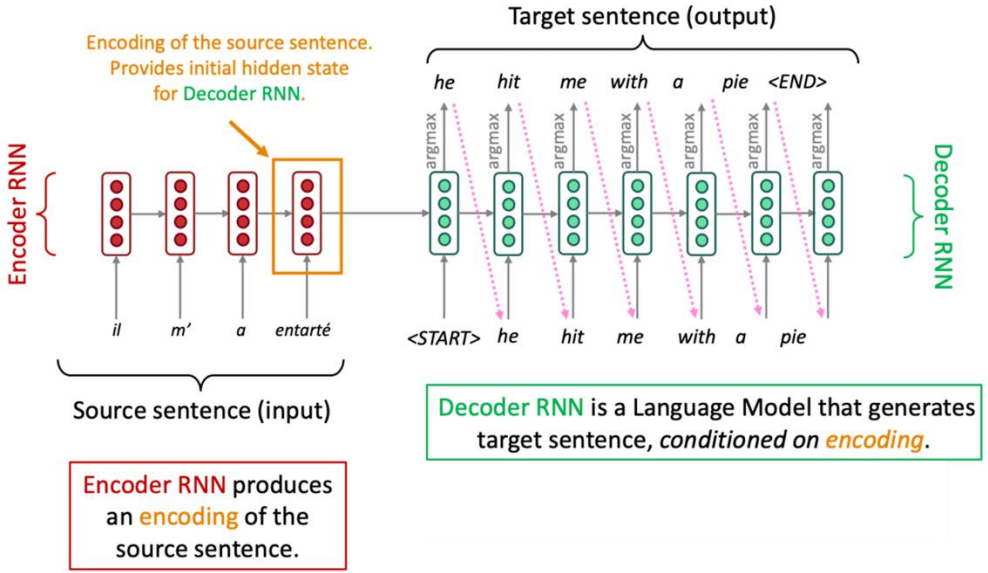
- Systems had many separately-designed subcomponents
  - Lots of feature Engineering
  - Required compiling and maintaining extra resources
  - Lots of human effort to maintain

Machine Translation - 2

❖ Neural Machine Translation

- A way to do Machine Translation with a single end-to-end neural network (using two RNNs)
- This neural network architecture is called a sequence-to-sequence model (aka seq2seq)
- An encoder-decoder model
  - One takes input and produces a neural representation
  - Another produces output based on that neural representation

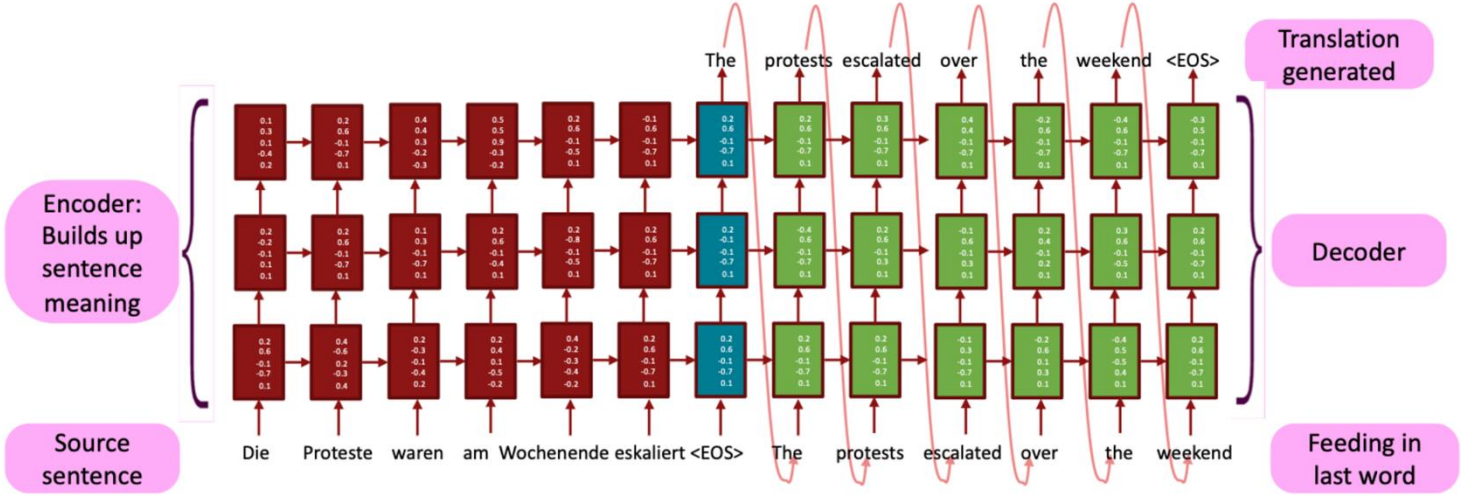
Conditional Language Model



Machine Translation - 3

❖ Multi-layer Deep Encoder-Decoder Machine Translation Net

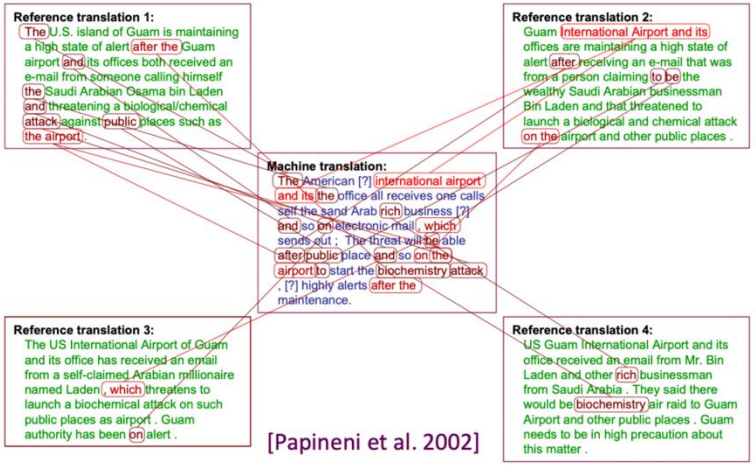
- 2-4 layers is best for encoder RNN and 4 layers for decoder
- Usually skip-connections/deeps-connections are needed to train deeper RNNs



Machine Translation Evaluation

❖ BiLingual Evaluation Understudy (BLEU)

- Compare the machine-written translation to one or several human-written translation(s)
- Compute a similarity score based on:
  - Geometric mean of n-gram precision (usually for 1,2,3 and **4-grams**)
  - Plus a penalty for too-short system translations
  - The more overlap, the better
- Useful but imperfect
  - Many valid ways to translate a sentence
  - A good translation can get a poor BLEU score
    - If it has low n-gram overlap with the human translation



Natural Language Processing with Deep Learning

CS224N

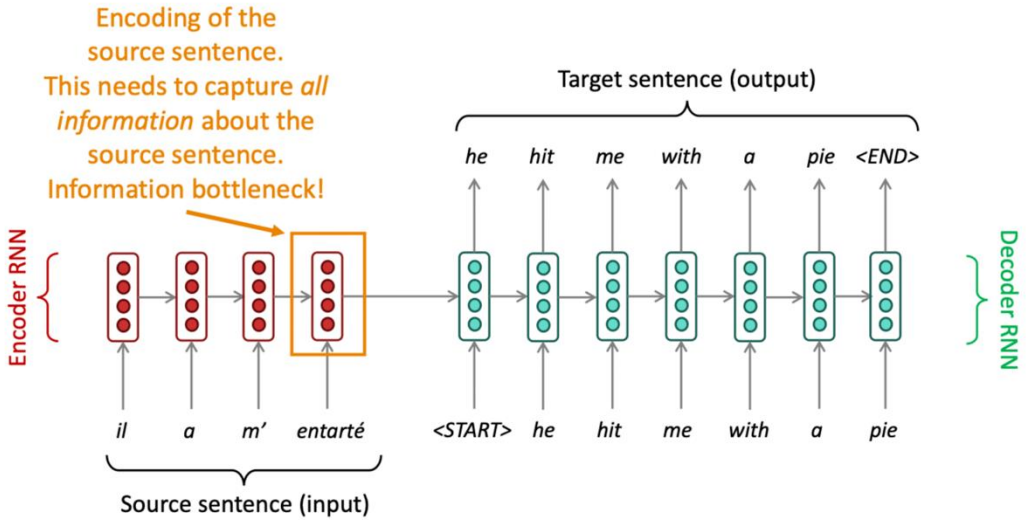
Lecture 7: Attention

@ CUAI Study

Problem in seq2seq Model

❖ Bottleneck problem

- General: A point in a process where the flow of information, data, or materials is limited or slowed down
- Everything useful about the sentence has to be stuffed into that one vector



Sequence-to-sequence with Attention

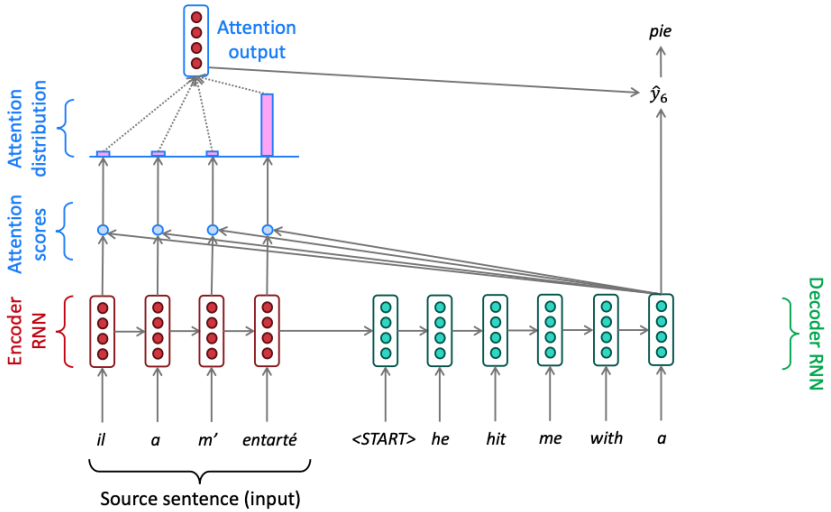
❖ Attention: in Equation

- Core-idea:
  - ➢ On each step of the decoder, use direct connection to the encoder to focus on a particular part of source sequence
- Encoder hidden states $h_1, \ldots, hn \in \mathbb{R}^h$
- Decoder hidden state $s_t \in \mathbb{R}^h$
- Attention scores: $e^t = [s_t^T h_1, \ldots, s_t^T h_n] \in \mathbb{R}^N$
- Processed as in the non-attention seq2seq model

$$\alpha^t = softmax(e^t) \in \mathbb{R}^N$$

$$a_t = \sum_{i=1}^{N} \alpha_i^t \, h_i \in \mathbb{R}^h$$

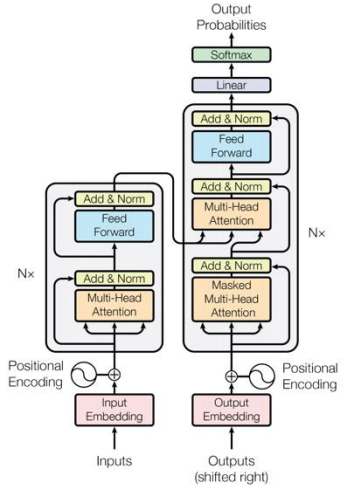
$$[a_t; s_t] \in \mathbb{R}^{2h}$$

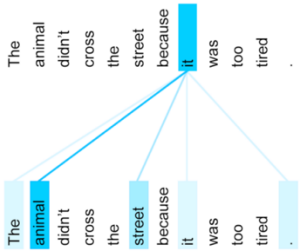


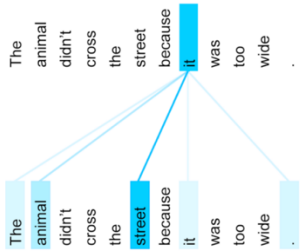
Attention is great!

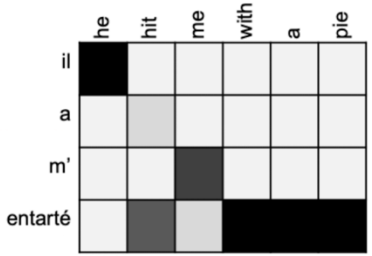
❖ Why attention is great?

- Significantly improve NMT performance
- Provide a more “human-like” model of the MT process
- Solve the bottleneck problem
- Help with the vanishing gradient problem
- Provide some interpretability









Thank You!

HTET ARKAR (hak3601@cau.ac.kr)