CS447: Natural Language Processing

http://courses.engr.illinois.edu/cs447

Lecture 14: Attention Mechanisms

Julia Hockenmaier

juliahmr@illinois.edu

3324 Siebel Center

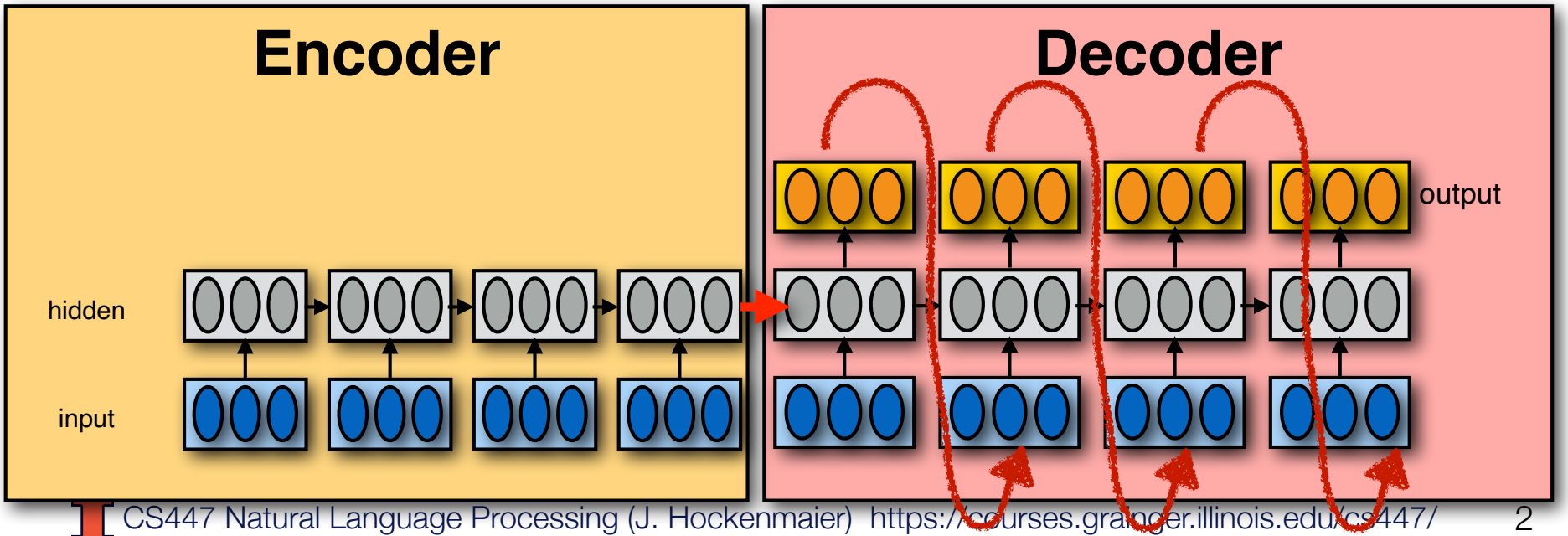
Encoder-Decoder (seq2seq) model

Task: Read an input sequence
and return an output sequence

- Machine translation: translate source into target language
- Dialog system/chatbot: generate a response

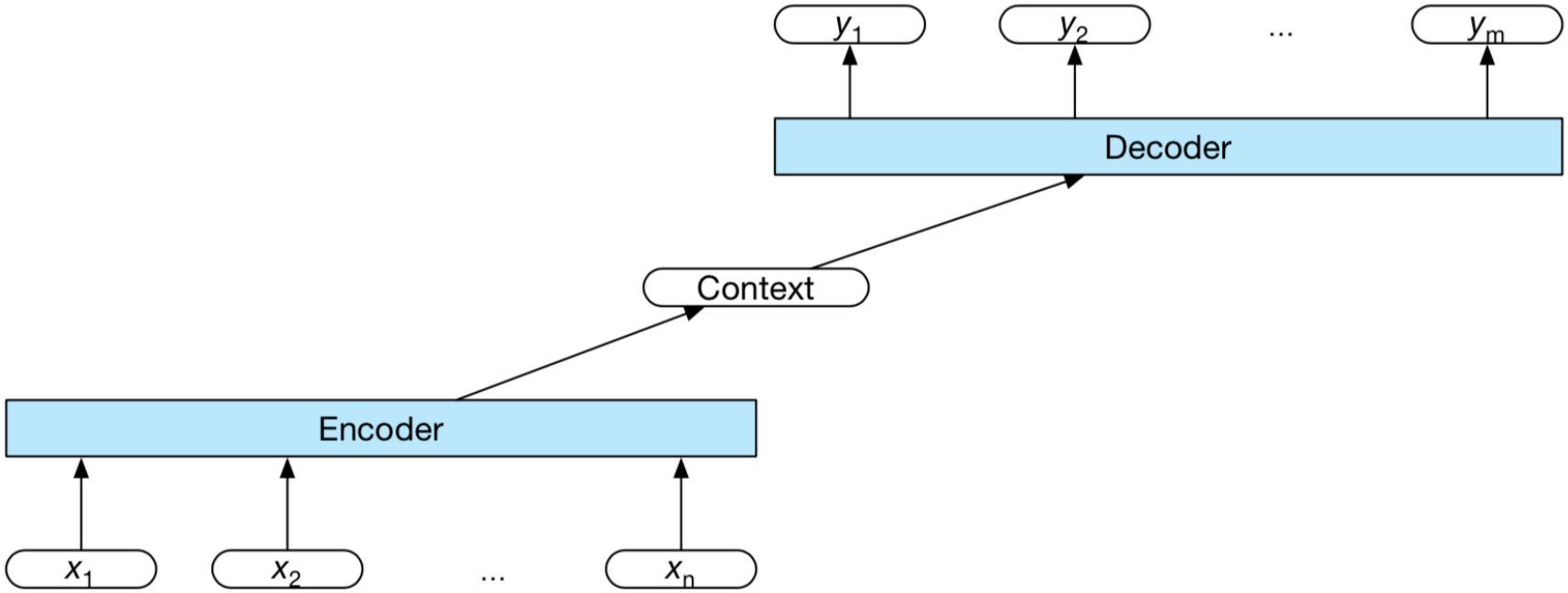
Reading the input sequence: **RNN Encoder**

Generating the output sequence: **RNN Decoder**



A more general view of seq2seq

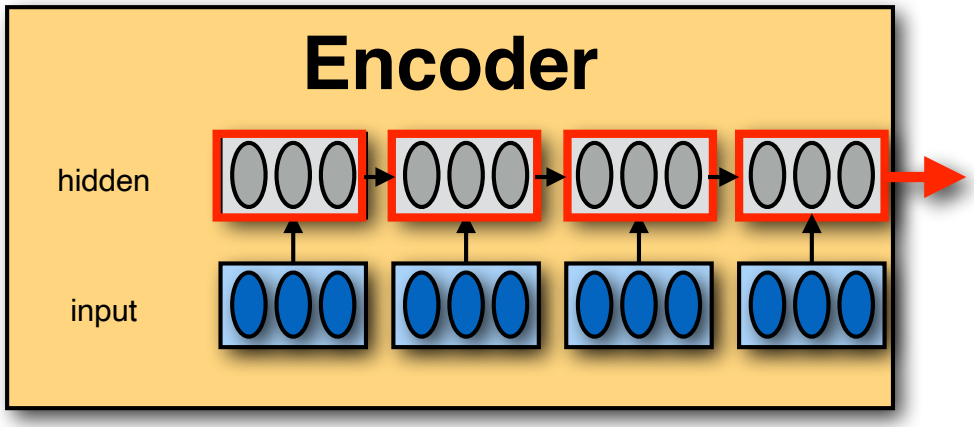
Insight 1: Any function of the encoder's output can be used as a **representation of the context** we want to condition the decoder on.



Encoder-Decoder (seq2seq) model

The **encoder** computes various representations of the input sequence:

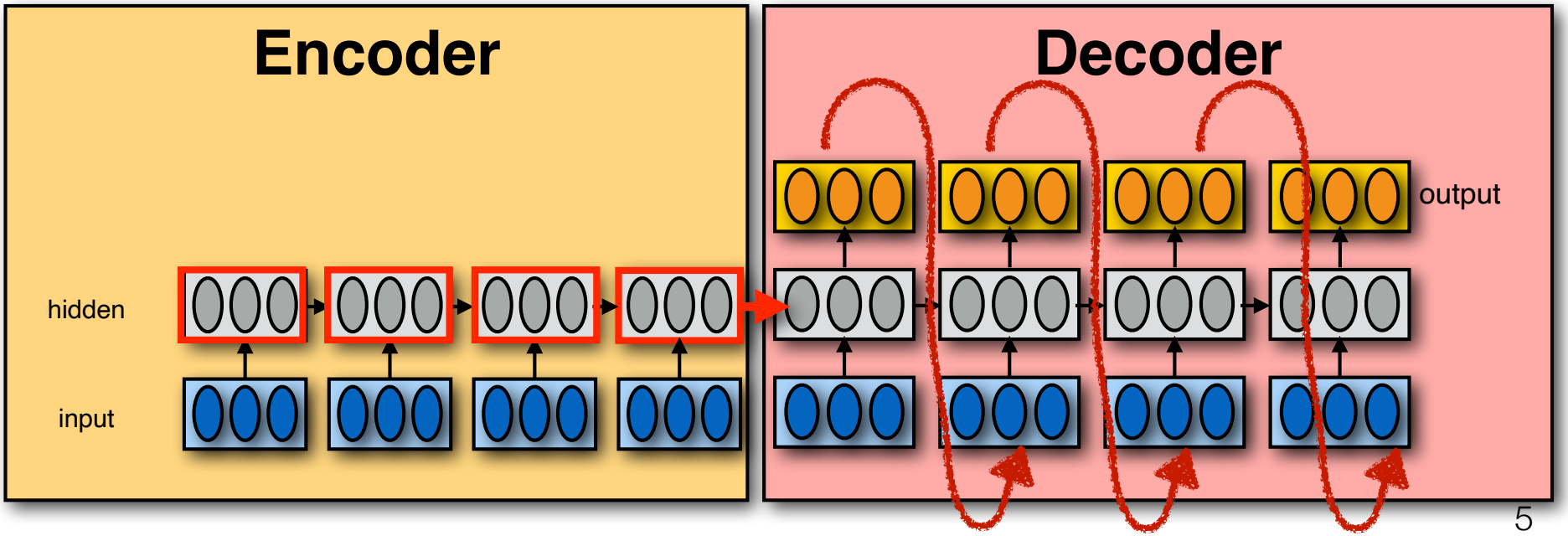
- The **last hidden state** of the encoder captures the entire sequence
- The **hidden state/output of each encoder element** captures a representation of the input sequence up to that input token



Encoder-Decoder (seq2seq) model

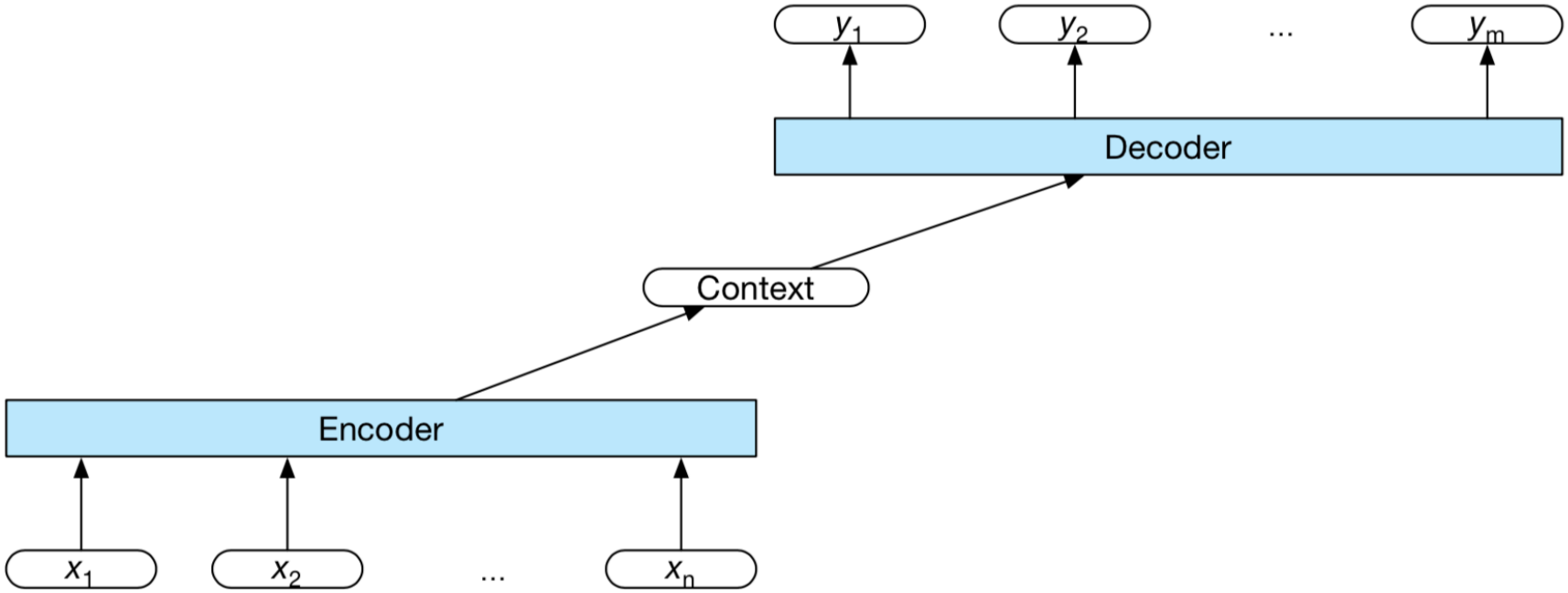
The **decoder** is a language model that generates an output sequence **conditioned on the input** sequence.

- **Vanilla RNN**: condition on the **last** hidden state
- **Attention**: condition on **all** hidden states



A more general view of seq2seq

Insight 1: Any function of the encoder's output can be used as a **representation of the context** we want to condition the decoder on.



Insight 2: We can feed the context in **at any time step** during decoding (not just at the beginning).

Representing context/history

The encoder maps each input element to a vector (the corresponding hidden state)

Attention allows the decoder to "look at" the encoder hidden states.

Challenge:

Which encoder states should the decoder consider?

How can multiple encoder states be combined into a **fixed-sized representation**?

Adding attention to the decoder

Basic idea: Feed a *d*-dimensional representation of the entire (arbitrary-length) input sequence into the decoder *at each time step during decoding.*

This representation of the input can be a **weighted average of the encoder's representation of the input** (i.e. hidden states)

The **averaging weights** associated with each encoder element specify how much attention to pay to that element

Since different parts of the input may be more or less important for different parts of the output, we want to **vary the weights** over the input during the decoding process.

(Cf. Word alignments in machine translation)

Adding attention to the decoder

We want to **condition the output** generation of the decoder on a **context-dependent representation of the input** sequence.

Attention computes a probability **distribution over the encoder's hidden states** that depends on the **decoder's current hidden state**

(This distribution is **computed anew for each output symbol**)

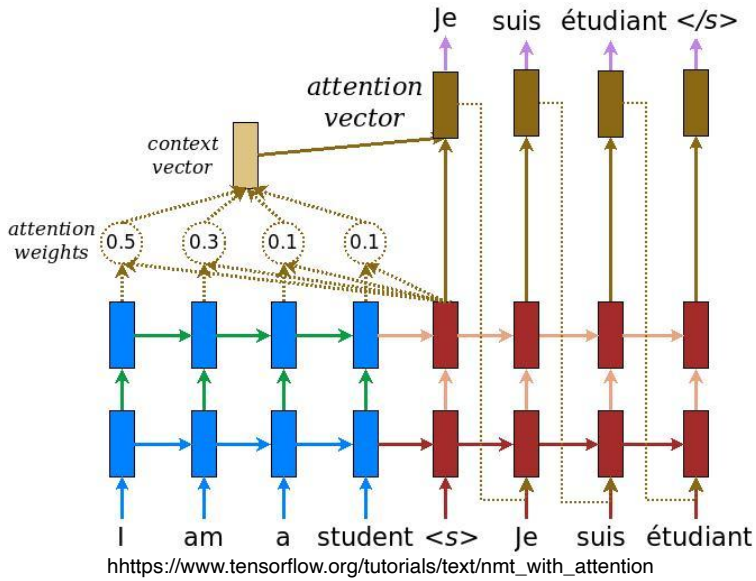
This attention distribution is used to compute **a weighted average of the encoder's hidden state vectors**.

This **context-dependent embedding of the input sequence** is fed into the output of the decoder RNN.

Attention, more formally

Define a **probability distribution** $\alpha^{(t)} = \left(\alpha_1^{(t)}, \ldots, \alpha_S^{(t)}\right)$ over the ***S* elements of the input** sequence **that depends on the current output** element *t*

Use this distribution to compute a **weighted average of the encoder's output** $\sum_{s=1..S} \alpha_s^{(t)} \mathbf{o}_s$ or hidden states $\sum_{s=1..S} \alpha_s^{(t)} \mathbf{h}_s$ and feed that into the decoder.



hhttps://www.tensorflow.org/tutorials/text/nmt_with_attention

Attention, more formally

1. Compute **a probability distribution** $\alpha^{(t)} = \left(\alpha_1^{(t)}, \ldots, \alpha_S^{(t)}\right)$ over the *encoder's* hidden states $\mathbf{h}^{(s)}$ that depends on the *decoder's* current $\mathbf{h}^{(t)}$

$$\alpha_s^{(t)} = \frac{\exp\left(s(\mathbf{h}^{(t)}, \mathbf{h}^{(s)})\right)}{\sum_{s'} \exp\left(s(\mathbf{h}^{(t)}, \mathbf{h}^{(s')})\right)}$$

2. Use $\alpha^{(t)}$ to compute **a weighted avg.** $\mathbf{c}^{(t)}$ of the *encoder's* $\mathbf{h}^{(s)}$:

$$\mathbf{c}^{(t)} = \sum_{s=1..S} \alpha_s^{(t)} \mathbf{h}^{(s)}$$

3. Use both $\mathbf{c}^{(t)}$ and $\mathbf{h}^{(t)}$ to compute **a new output** $\mathbf{o}^{(t)}$, e.g. as

$$\mathbf{o}^{(t)} = \tanh\left(W_1 \mathbf{h}^{(t)} + W_2 \mathbf{c}^{(t)}\right)$$

Defining Attention Weights

Hard attention (degenerate case, non-differentiable):

$\alpha^{(t)} = \left(\alpha_1^{(t)}, \ldots, \alpha_S^{(t)}\right)$ is a **one-hot vector**

(e.g. 1 = most similar element to decoder's vector, 0 = all other elements)

Soft attention (general case):

$\alpha^{(t)} = \left(\alpha_1^{(t)}, \ldots, \alpha_S^{(t)}\right)$ is **not a one-hot**

- Use the **dot product** (no learned parameters):

$$s\left(\mathbf{h}^{(t)}, \mathbf{h}^{(s)}\right) = \mathbf{h}^{(t)} \cdot \mathbf{h}^{(s)}$$

- Learn a **bilinear matrix** W:

$$s\left(\mathbf{h}^{(t)}, \mathbf{h}^{(s)}\right) = \left(\mathbf{h}^{(t)}\right)^T W \mathbf{h}^{(s)}$$

- Learn **separate weights** for the hidden states:

$$s\left(\mathbf{h}^{(t)}, \mathbf{h}^{(s)}\right) = \mathbf{v}^T \tanh\left(W_1 \mathbf{h}^{(t)} + W_2 \mathbf{h}^{(s)}\right)$$