

AI504: Programming for Artificial Intelligence

Week 10: Recurrent Neural Network

Edward Choi

Grad School of AI

edwardchoi@kaist.ac.kr

Today's Topic

- Recurrent Neural Network
 - Vanilla RNN
 - Bidirectional RNN
 - GRU, LSTM
- Sequence-to-sequence
 - Neural Machine Translation
- Attention

Recurrent Neural Network

Handling Variable-Length Sequences

- Image-to-Label

VS

- Sentence-to-Label

Handling Variable-Length Sequences

- Image-to-Label ➔ Input size is fixed
- VS
- Sentence-to-Label ➔ Input size varies by sample

Bag-of-Words

- Classical way to handle variable length sentences/documents
- I gave the ball to John, who gave it to Mary
 - I:1, gave:2, the:1, ball:1, to:2, John:1, who:1, it:1, Mary:1

Bag-of-Words

- I gave the ball to John, who gave it to Mary
 - I:1, gave:2, the:1, ball:1, to:2, John:1, who:1, it:1, Mary:1

All vocab	Word count
a	0
aardvark	0
ab	0
...	...
...	...
ball	2
...	...
gave	2
...	...

Bag-of-Words

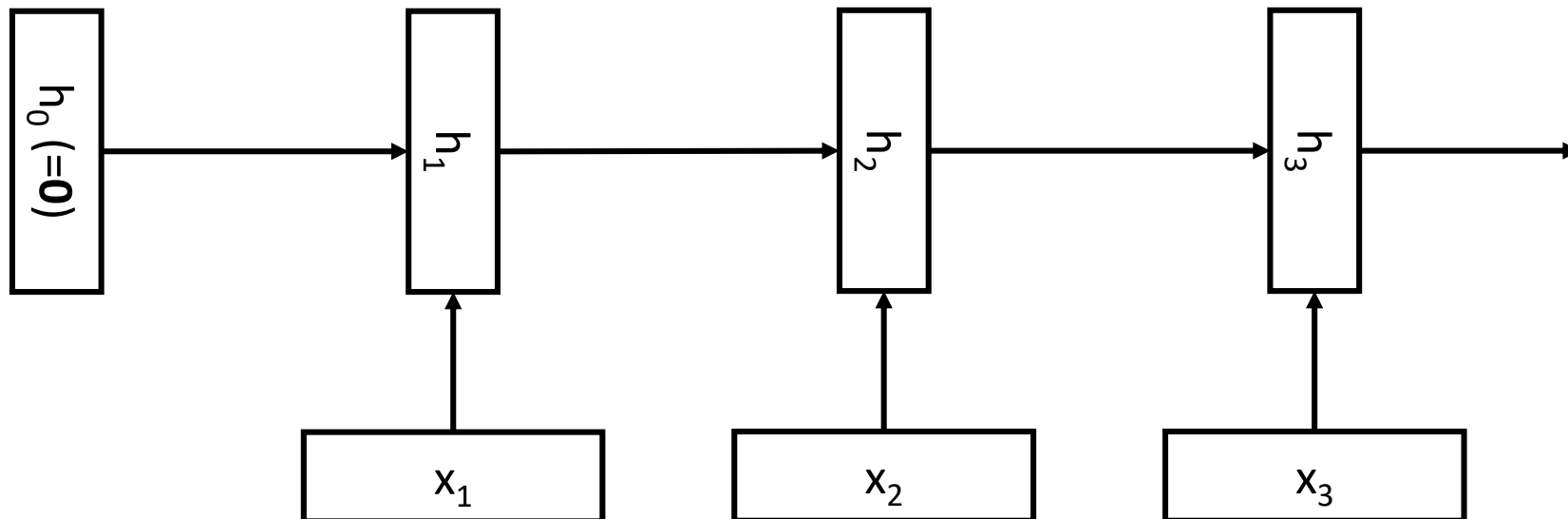
- I gave the ball to John, who gave it to Mary
 - I:1, gave:2, the:1, ball:1, to:2, John:1, who:1, it:1, Mary:1
- I gave the ball to Mary, who gave it to John
 - I:1, gave:2, the:1, ball:1, to:2, John:1, who:1, it:1, Mary:1
- Different meaning, same representation!

Classical NLP

- Syntax, Semantic, Discourse, Pragmatic
 - Part-of-speech tagging
 - Parsing
 - Named entity recognition
 - Semantic role labeling
-
- Many of them made obsolete by deep learning
 - Or are they...?

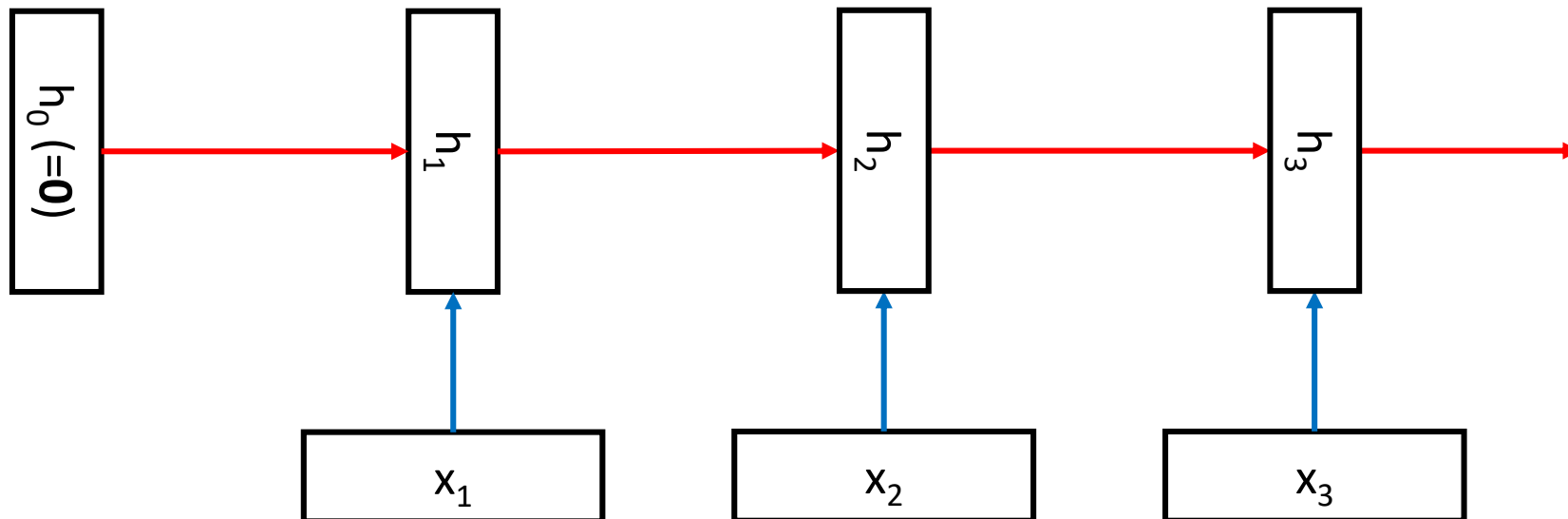
Recurrent Neural Network

- Represent variable-length input
- $\mathbf{h}_t = f(W\mathbf{x}_t + U\mathbf{h}_{t-1} + \mathbf{b})$
 - f : non-linear activation function (originally tanh)



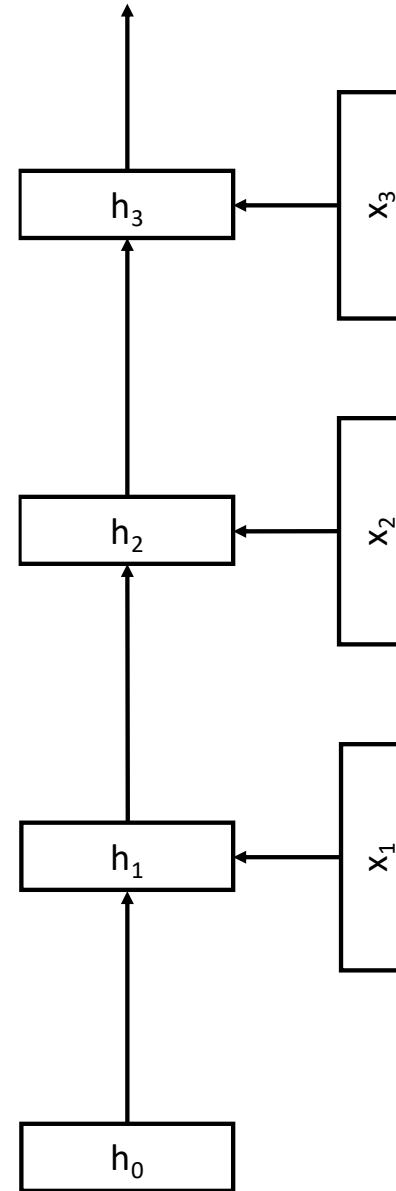
RNN

- Represent variable-length input
- Same weights at each timestep to handle variable-length sequence
 - U : h_{t-1} to h_t
 - W : x_t to h_t



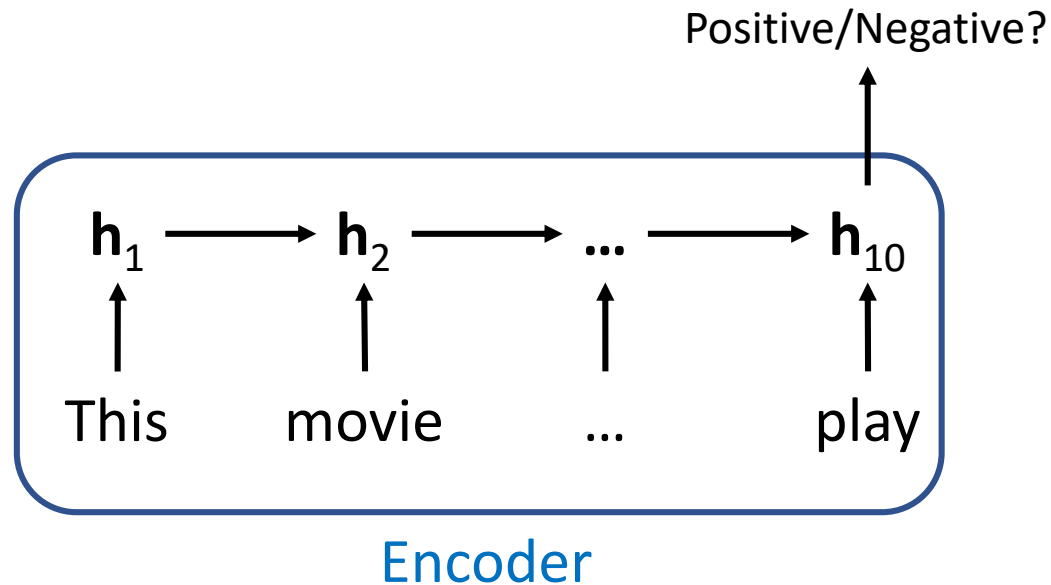
RNN

- Represent variable-length input
- Feedforward Neural Network with new information at each timestep.
 - But use the same weights repeatedly.



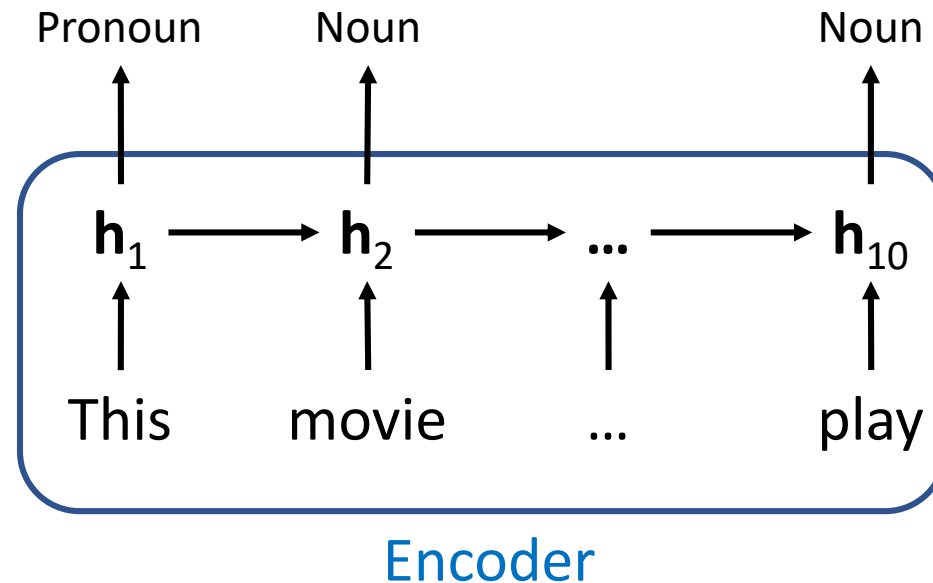
Application

- Sequence-level classification/regression
 - One RNN to encode input
 - One prediction
 - e.g. Sentiment classification, topic classification



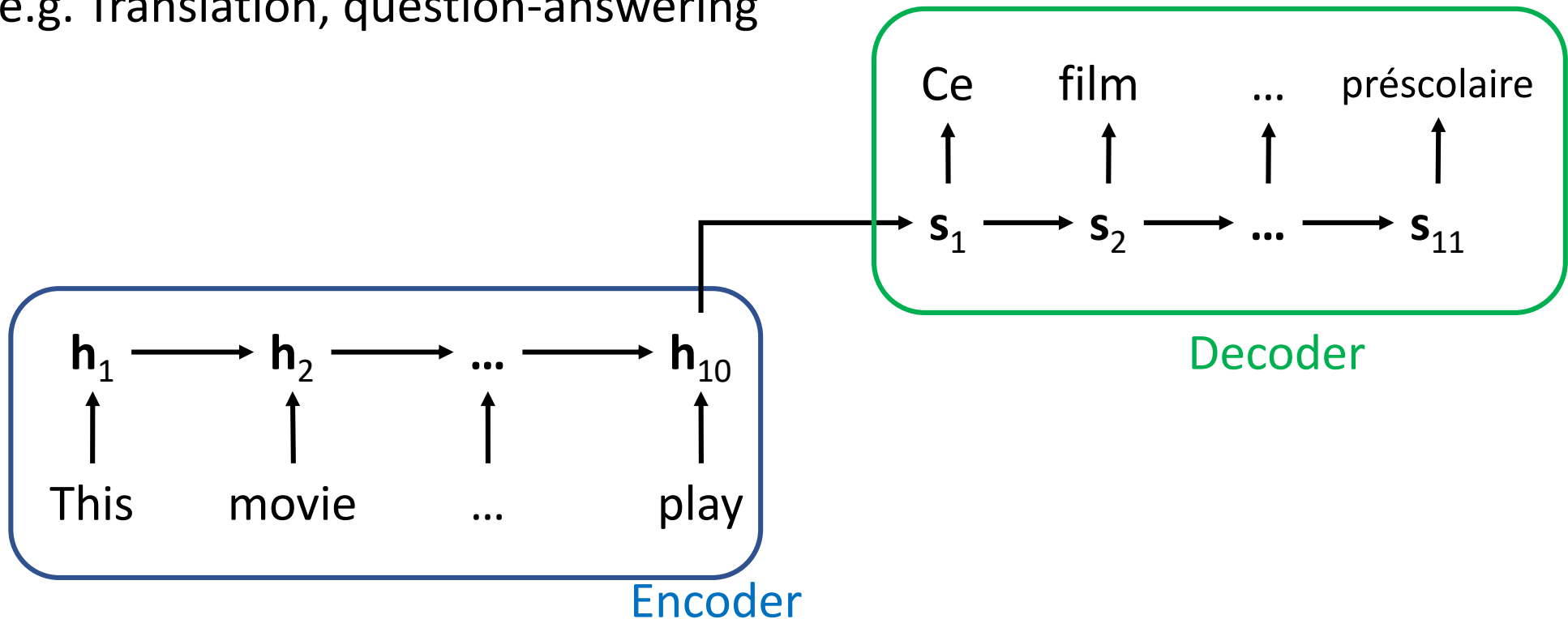
Application

- Token-level classification/regression
 - One RNN to encode input
 - Multiple predictions (as many as the number of tokens)
 - e.g. Part-of-speech tagging, language modeling



Application

- Many-to-many (Seq2seq)
 - One RNN to encode input
 - One RNN to decode output
 - e.g. Translation, question-answering

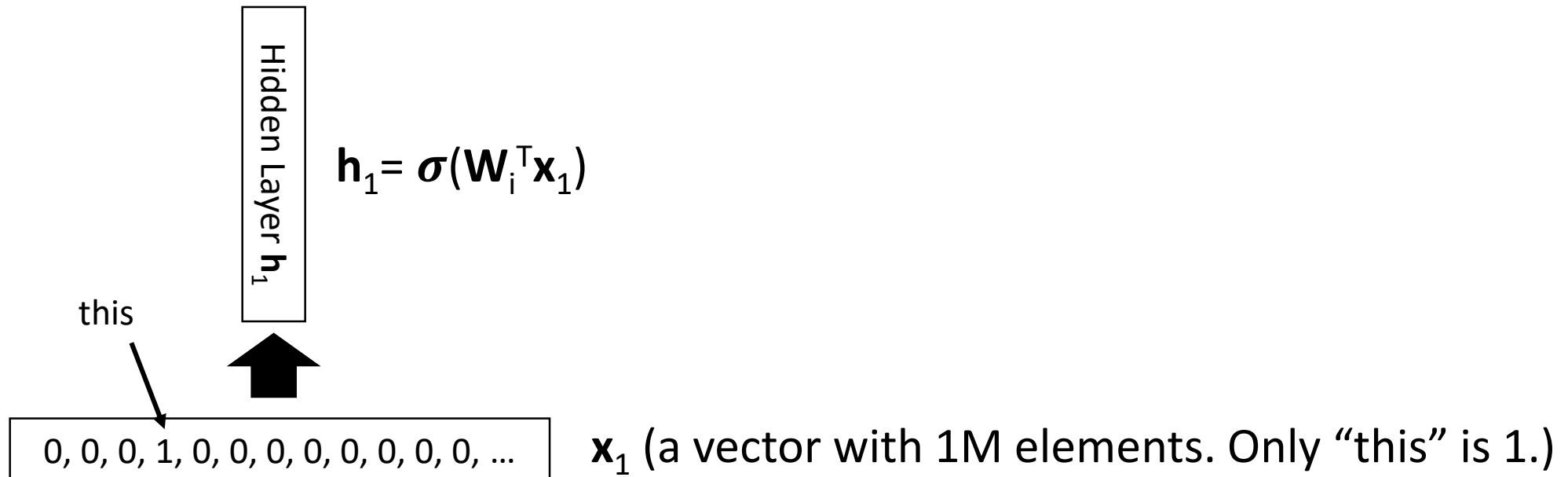


Sequence-level Classification

- Sentiment classification: **Positive** or **Negative**?
 - “This movie is as impressive as a preschool Christmas play”

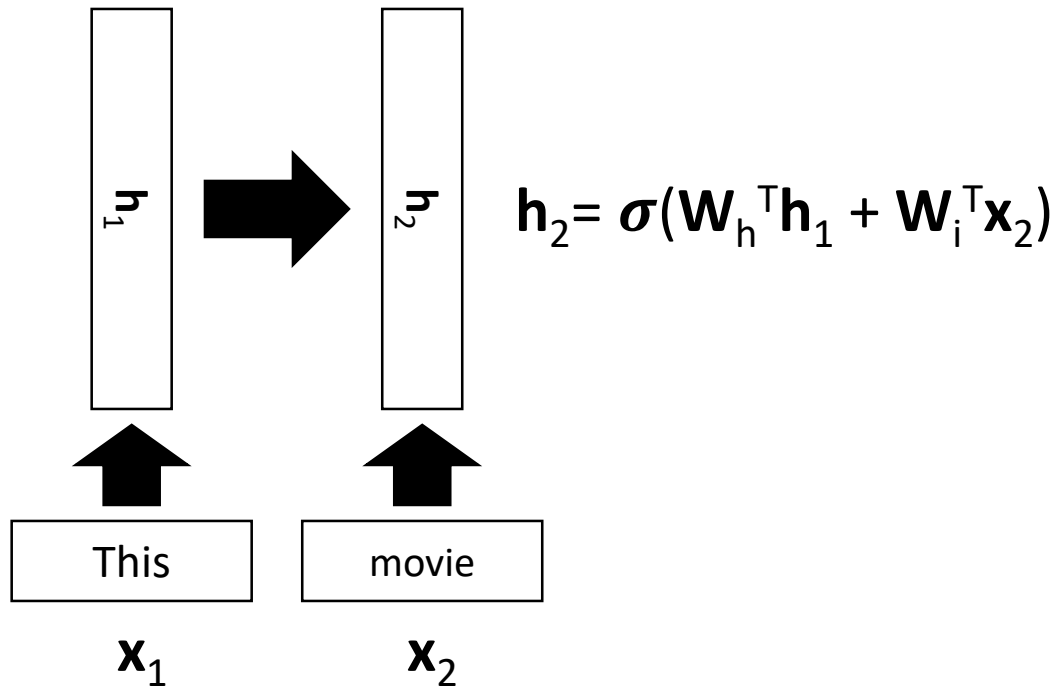
Sequence prediction with RNN

- Sentiment classification: **Positive** or **Negative**?
 - “This movie is as impressive as a preschool Christmas play”



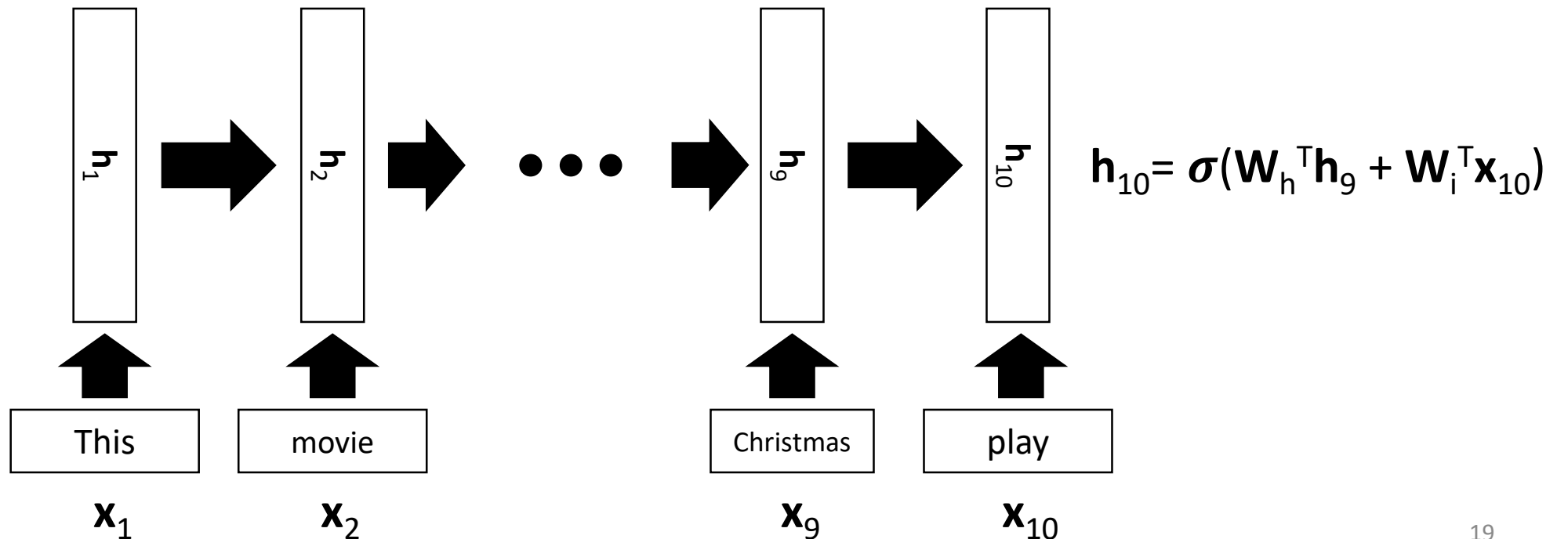
Sequence prediction with RNN

- Sentiment classification: **Positive** or **Negative**?
 - “This movie is as impressive as a preschool Christmas play”



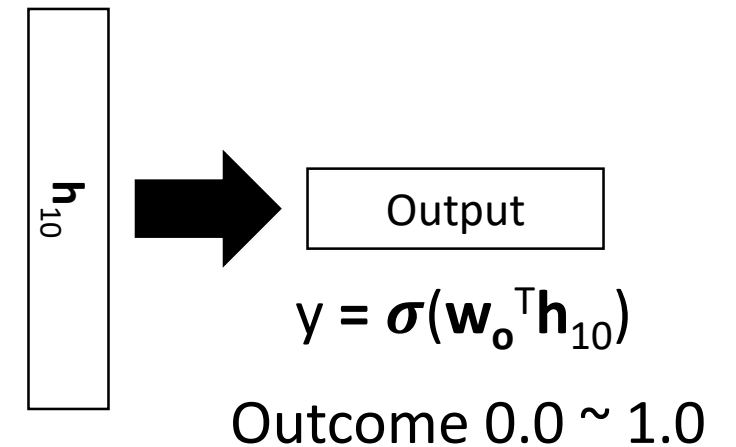
Sequence prediction with RNN

- Sentiment classification: **Positive** or **Negative**?
 - “This movie is as impressive as a preschool Christmas play”



Sequence prediction with RNN

- Sentiment classification: **Positive** or **Negative**?
 - “This movie is as impressive as a preschool Christmas play”



Language Modeling

- $p(\text{"This movie is as impressive as a preschool Christmas play"})$
 - What is the probability of this sentence?
 - (Probably super small...)

Language Modeling

- $p(\text{"This movie is as impressive as a preschool Christmas play"})$



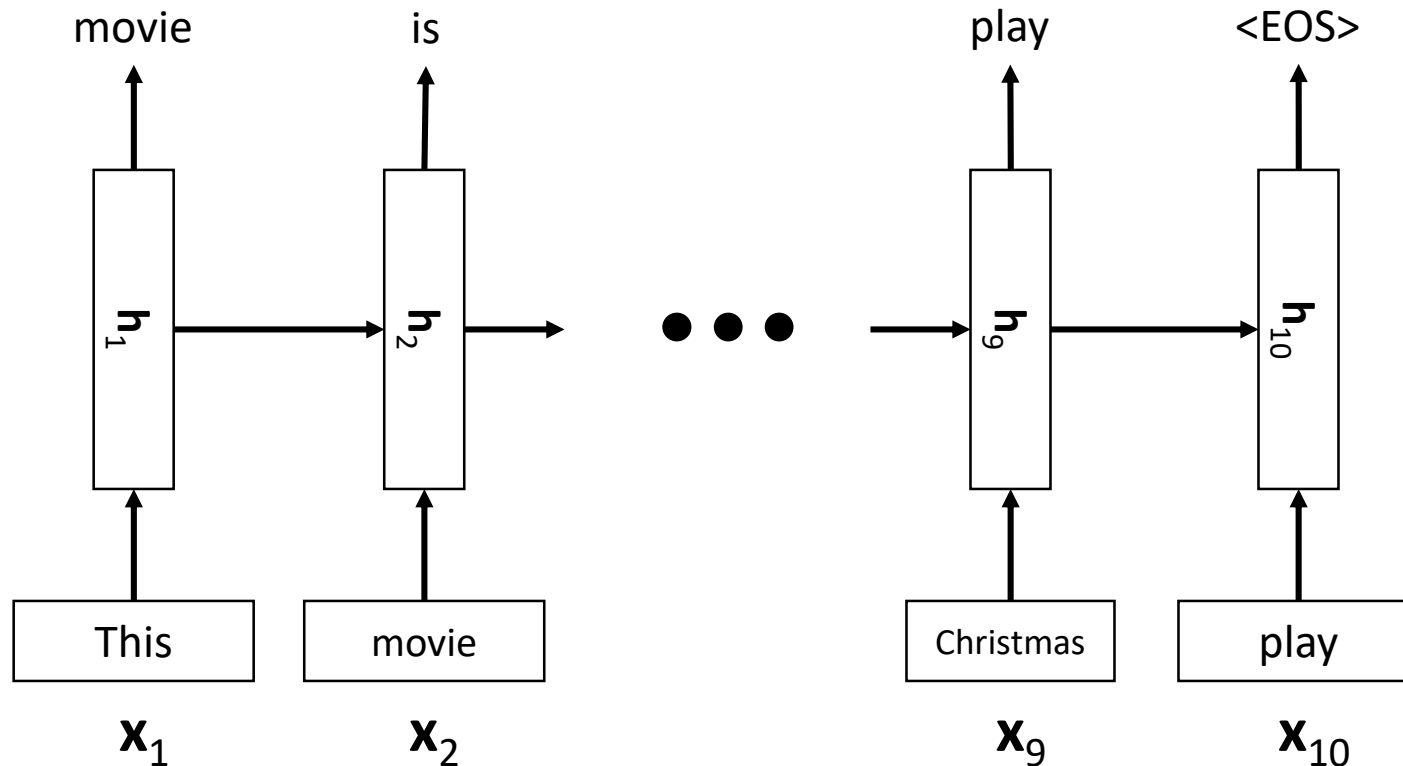
- $p(\text{This}) * p(\text{movie} \mid \text{This}) * p(\text{is} \mid \text{This, movie}) * \dots * p(\text{play} \mid \text{This, movie, ..., Christmas})$
- Need a model that can perform:
 - $p(w_t \mid w_1, w_2, \dots, w_{t-1})$

Language Modeling

- $p(\text{"This movie is as impressive as a preschool Christmas play"})$
 - $p(\text{This}) * p(\text{movie} \mid \text{This}) * p(\text{is} \mid \text{This, movie}) * \dots * p(\text{play} \mid \text{This, movie, ..., Christmas})$
- Need a model that can perform:
 - $p(w_t \mid w_1, w_2, \dots, w_{t-1})$
- Traditionally:
 - Unigram, bigram, trigram
 - Bigram $\Rightarrow p(w_t \mid w_{t-1})$, Trigram $\Rightarrow p(w_t \mid w_{t-2}, w_{t-1})$
 - Limited horizon
- With RNN
 - Theoretically, can model full $p(w_t \mid w_1, w_2, \dots, w_{t-1})$

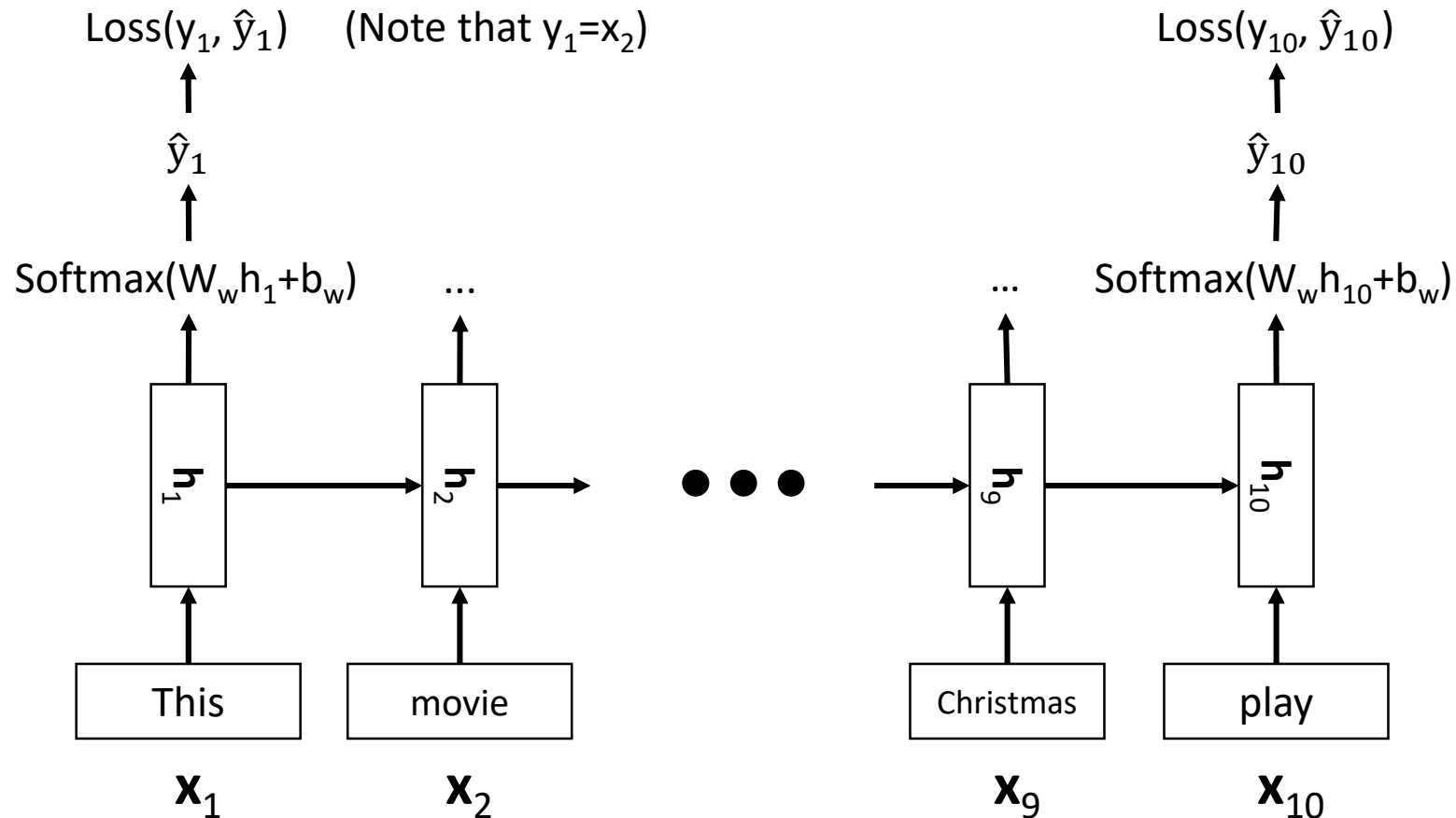
Language Modeling with RNN

- “This movie is as impressive as a preschool Christmas play”



Language Modeling with RNN

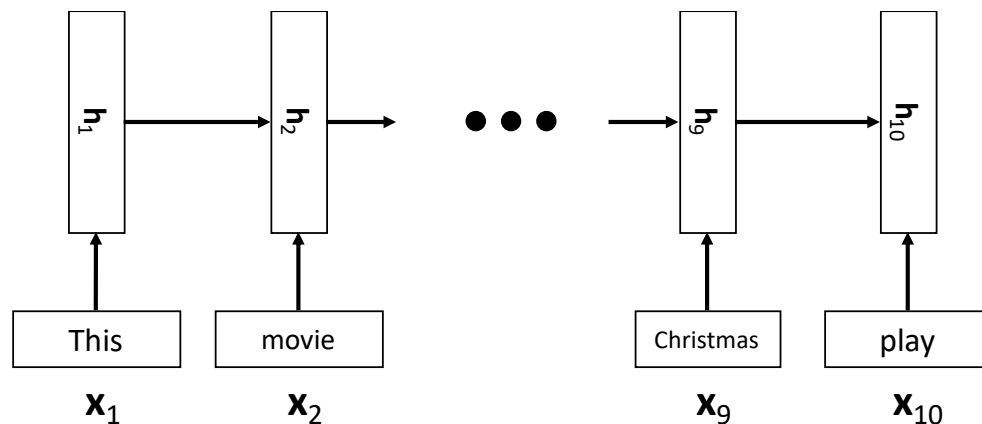
- “This movie is as impressive as a preschool Christmas play”



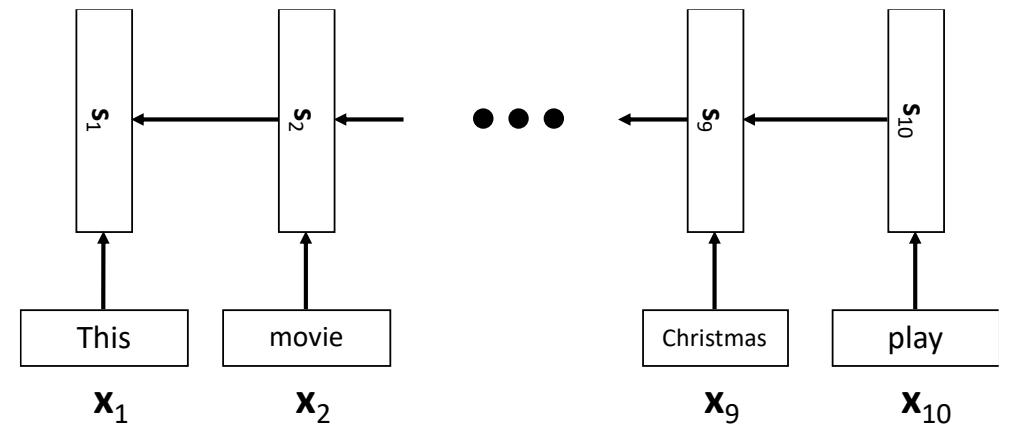
➡ 10 loss terms for a single sentence (Loss is usually negative log-likelihood)

Bidirectional RNN

- Encode a sequence in two directions



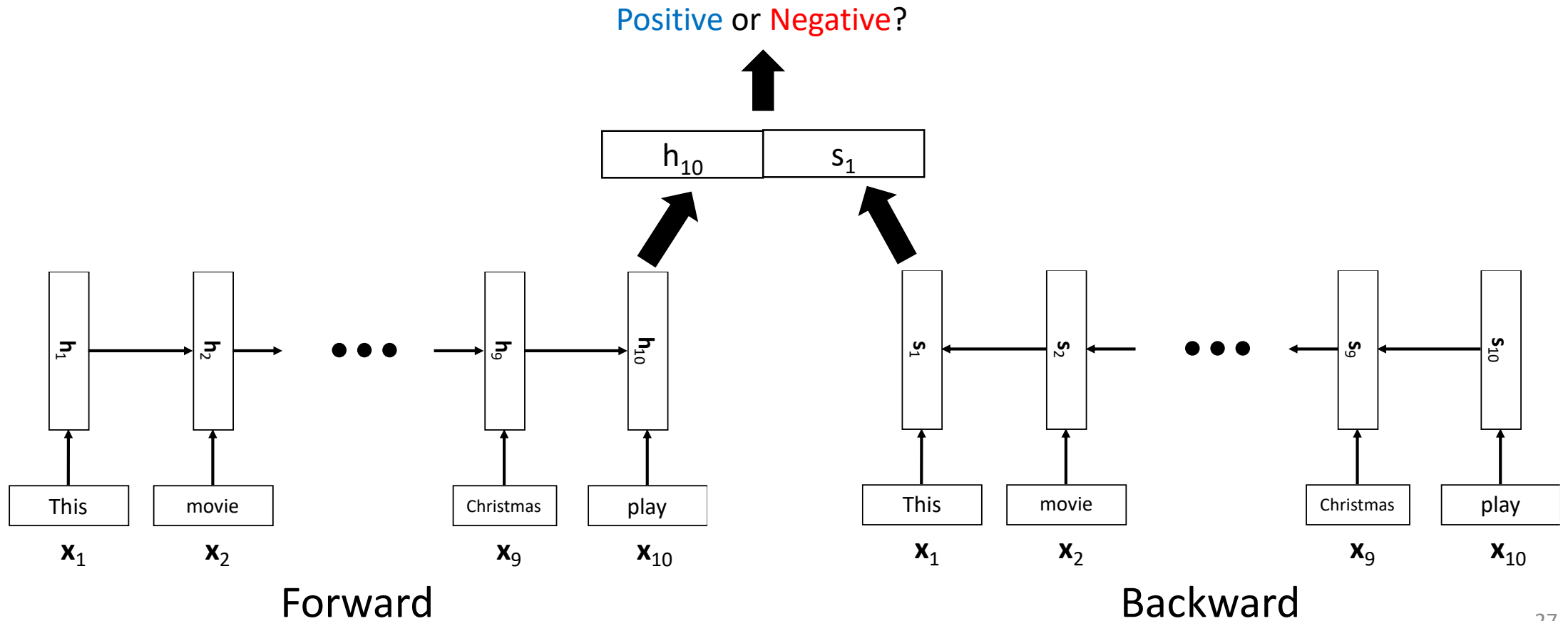
Forward



Backward

Bidirectional RNN

- Encode a sequence in two directions



Limitation

- Vanishing gradient still exists.
 - Long sequence means long backpropagation chain!
- Input from a distant past is forgotten!
 - Ex: “Jane walked into the room. John walked in too. It was late in the day. Jane said hi to _____”
- How to remedy this?
 - Some old tricks: Initialize weight matrices to identity matrices, use ReLU.
- Exploding gradient also exists.
 - Popular remedy: gradient clipping

Gated Recurrent Unit

- More complex hidden unit computation in recurrence!
- Gated Recurrent Units (GRU) introduced by Cho et al. 2014 (see reading list)
- Main ideas:
 - keep around memories to capture long distance dependencies
 - allow error messages to flow at different strengths depending on the inputs

Gated Recurrent Unit

- Standard RNN computes hidden layer at next time step directly:

$$h_t = f \left(W^{(hh)} h_{t-1} + W^{(hx)} x_t \right)$$

- GRU first computes an update **gate** (another layer) based on current input word vector and hidden state

$$z_t = \sigma \left(W^{(z)} x_t + U^{(z)} h_{t-1} \right)$$

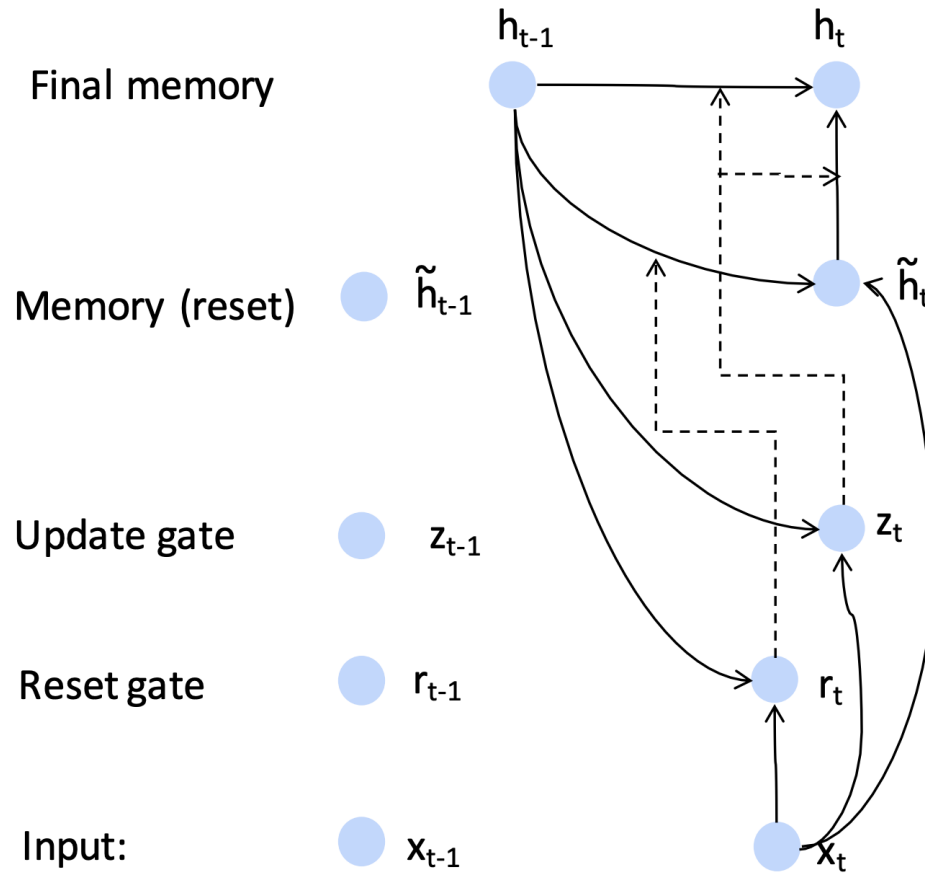
- Compute reset gate similarly but with different weights

$$r_t = \sigma \left(W^{(r)} x_t + U^{(r)} h_{t-1} \right)$$

Gated Recurrent Unit

- Update gate $z_t = \sigma \left(W^{(z)} x_t + U^{(z)} h_{t-1} \right)$
- Reset gate $r_t = \sigma \left(W^{(r)} x_t + U^{(r)} h_{t-1} \right)$
- New memory content: $\tilde{h}_t = \tanh (W x_t + r_t \circ U h_{t-1})$
If reset gate unit is ~ 0 , then this ignores previous memory and only stores the new word information
- Final memory at time step combines current and previous time steps: $h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t$

Gated Recurrent Unit



$$z_t = \sigma \left(W^{(z)} x_t + U^{(z)} h_{t-1} \right)$$

$$r_t = \sigma \left(W^{(r)} x_t + U^{(r)} h_{t-1} \right)$$

$$\tilde{h}_t = \tanh \left(W x_t + r_t \circ U h_{t-1} \right)$$

$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \tilde{h}_t$$

Long Short Term Memory

- We can make the units even more complex
- Allow each time step to modify
 - Input gate (current cell matters) $i_t = \sigma \left(W^{(i)} x_t + U^{(i)} h_{t-1} \right)$
 - Forget (gate 0, forget past) $f_t = \sigma \left(W^{(f)} x_t + U^{(f)} h_{t-1} \right)$
 - Output (how much cell is exposed) $o_t = \sigma \left(W^{(o)} x_t + U^{(o)} h_{t-1} \right)$
 - New memory cell $\tilde{c}_t = \tanh \left(W^{(c)} x_t + U^{(c)} h_{t-1} \right)$
- Final memory cell: $c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$
- Final hidden state: $h_t = o_t \circ \tanh(c_t)$

Sequence-to-Sequence

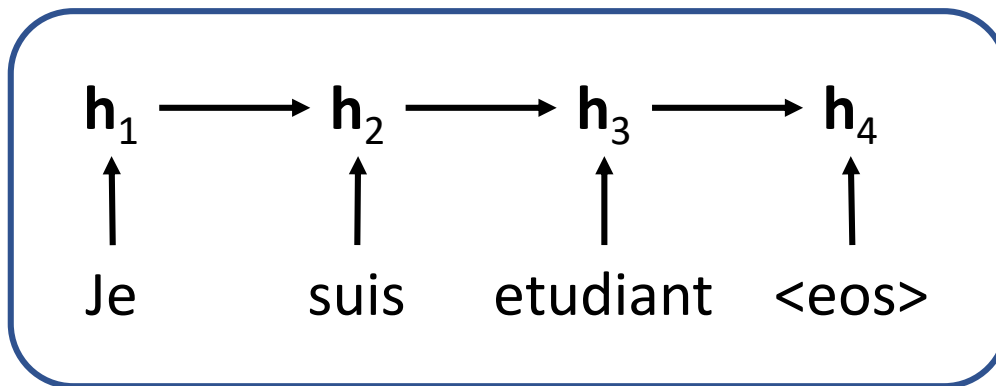
Sequence-to-Sequence

- Given **variable-length sequence input**,
Predict (Generate) **variable-length sequence output**
 - Machine translation
 - Question answering
 - Chatbot
- Naturally, we need two RNNs!
 - Why?

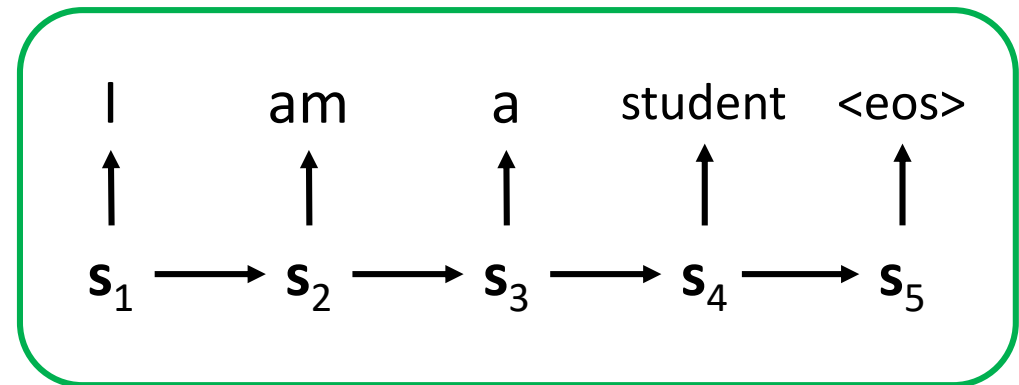
Machine Translation

- “Je suis etudiant” → “I am a student”
 - French-to-English

Encoder



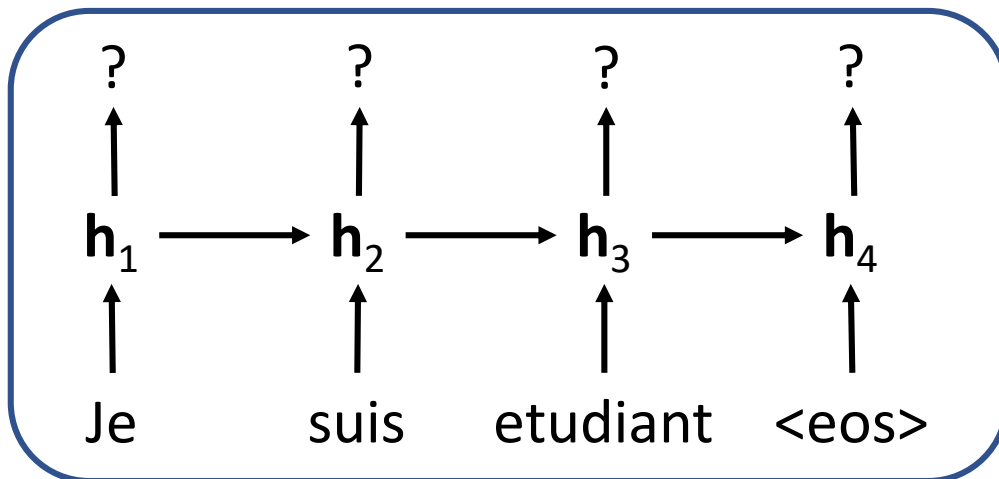
Decoder



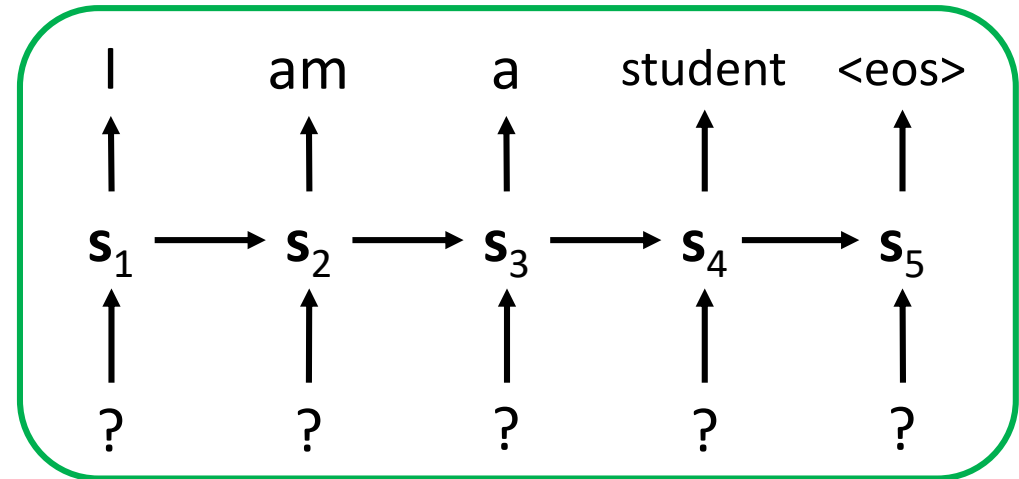
Machine Translation

- What is the output of Encoder?
- What is the input of Decoder?

Encoder

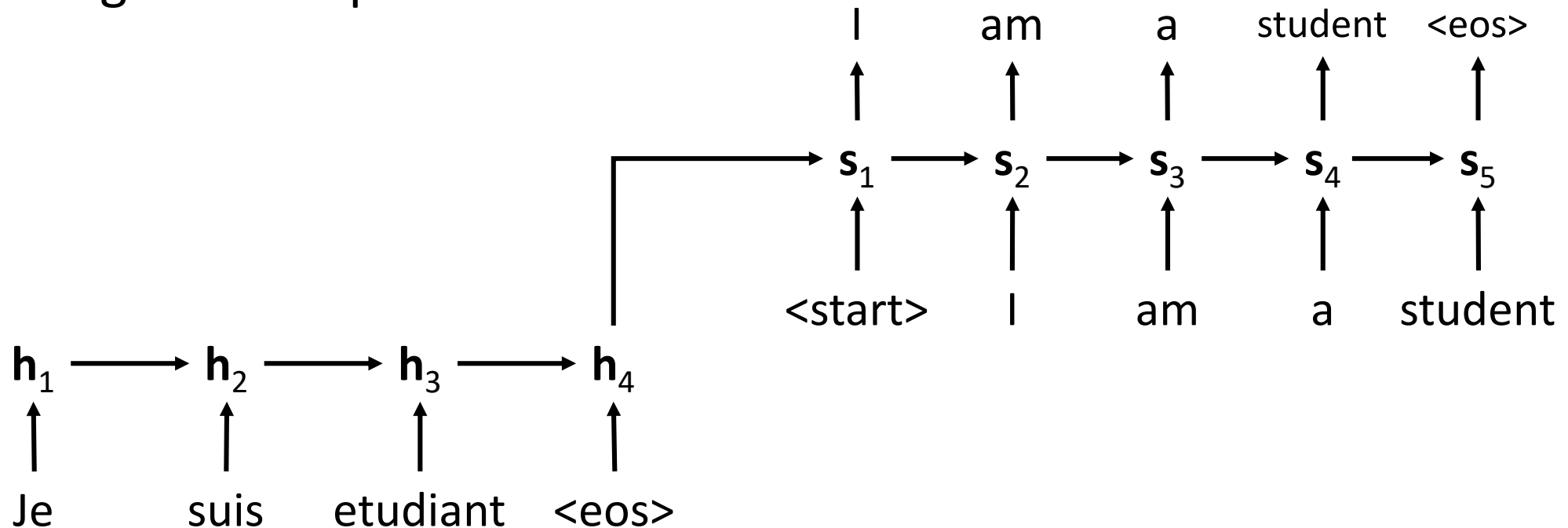


Decoder



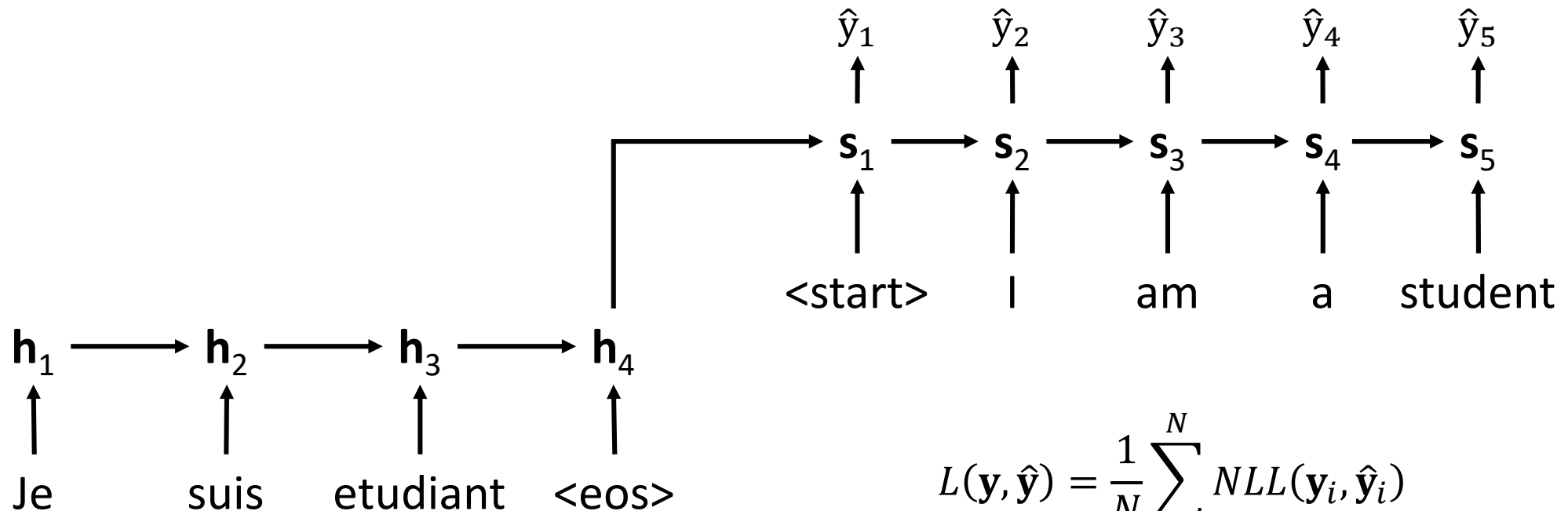
Machine Translation

- Encoder's last hidden layer is the initial state of Decoder
 - h_4 represents the input sentence
- Autoregressive input to Decoder



Machine Translation

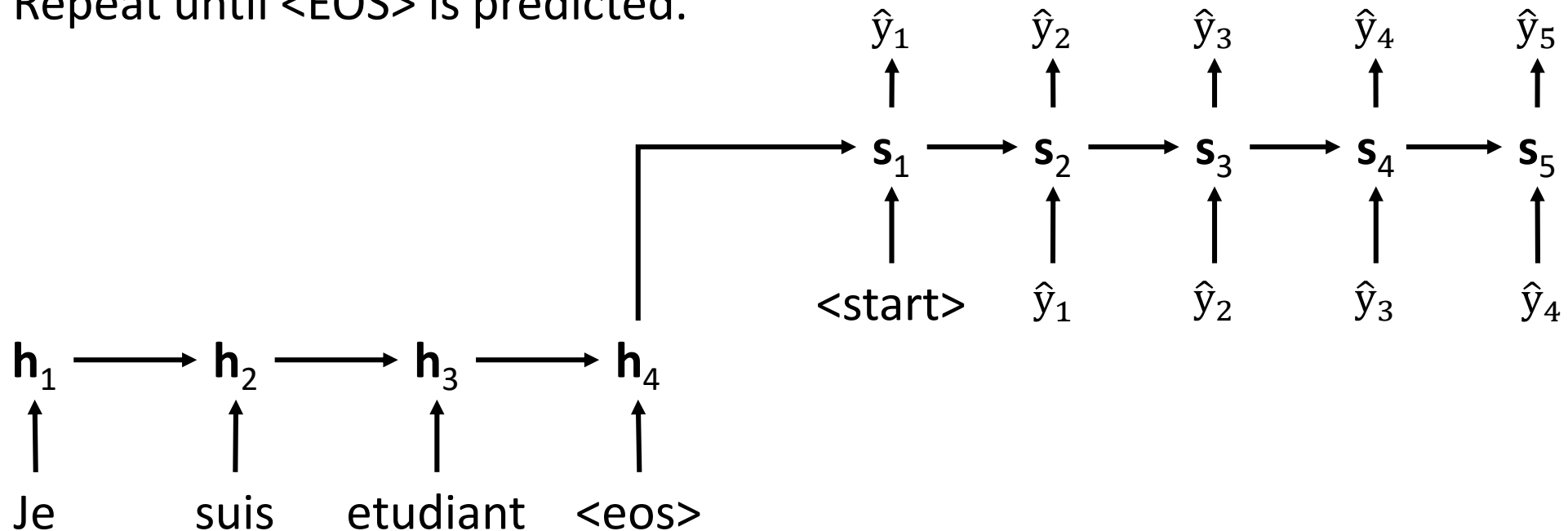
- At training phase:
 - Decoder input is the ground true tokens.
 - Apply negative log-likelihood to predicted outputs.



$$L(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{N} \sum_{i=1}^N NLL(\mathbf{y}_i, \hat{\mathbf{y}}_i)$$

Machine Translation

- At test phase:
 - Decoder input is the previous predicted token.
 - Autoregressive input
 - Repeat until <EOS> is predicted.



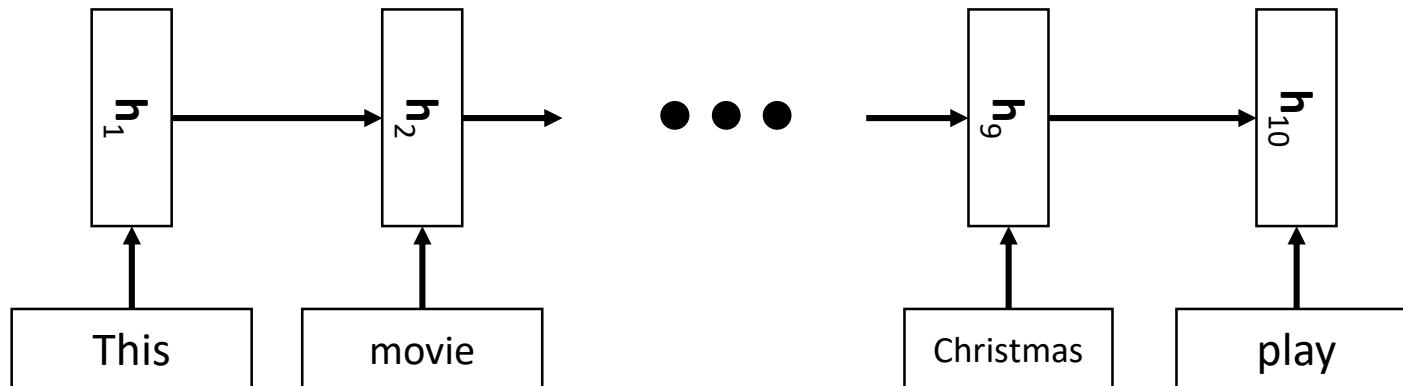
Attention

Attention models

- Bahdanau, Cho, Bengio, 2014
 - English-French translation using RNN
- Let's use hidden layers from all timesteps to make predictions

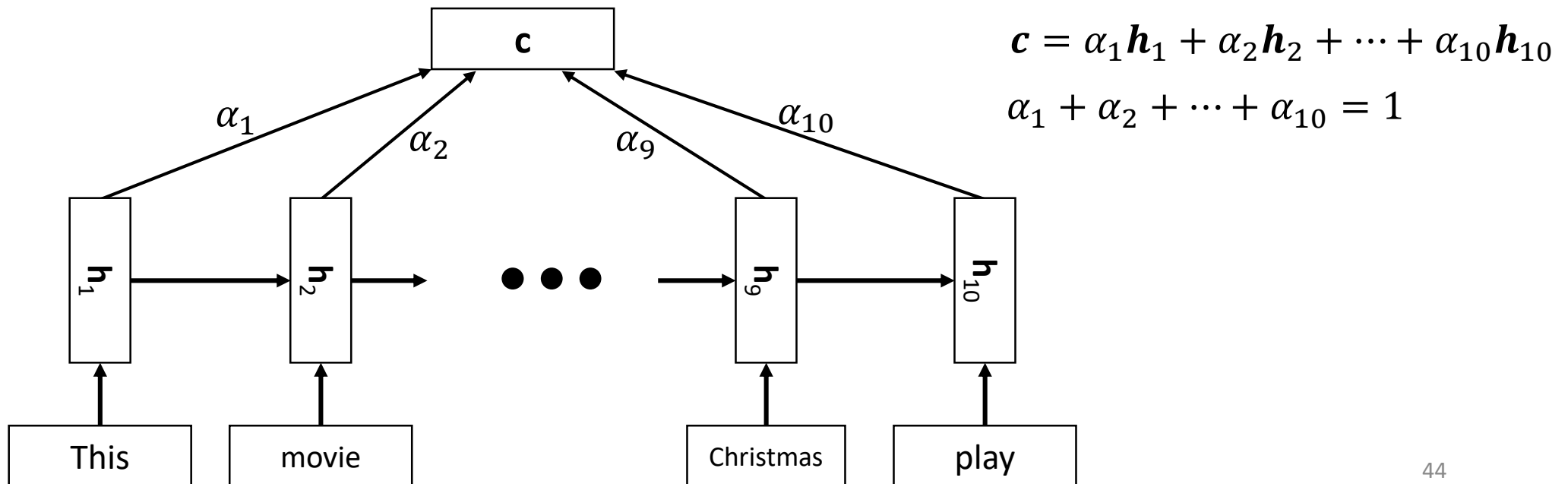
Attention models

- Bahdanau, Cho, Bengio, 2014
 - English-French translation using RNN
- Let's use hidden layers from all timesteps to make predictions



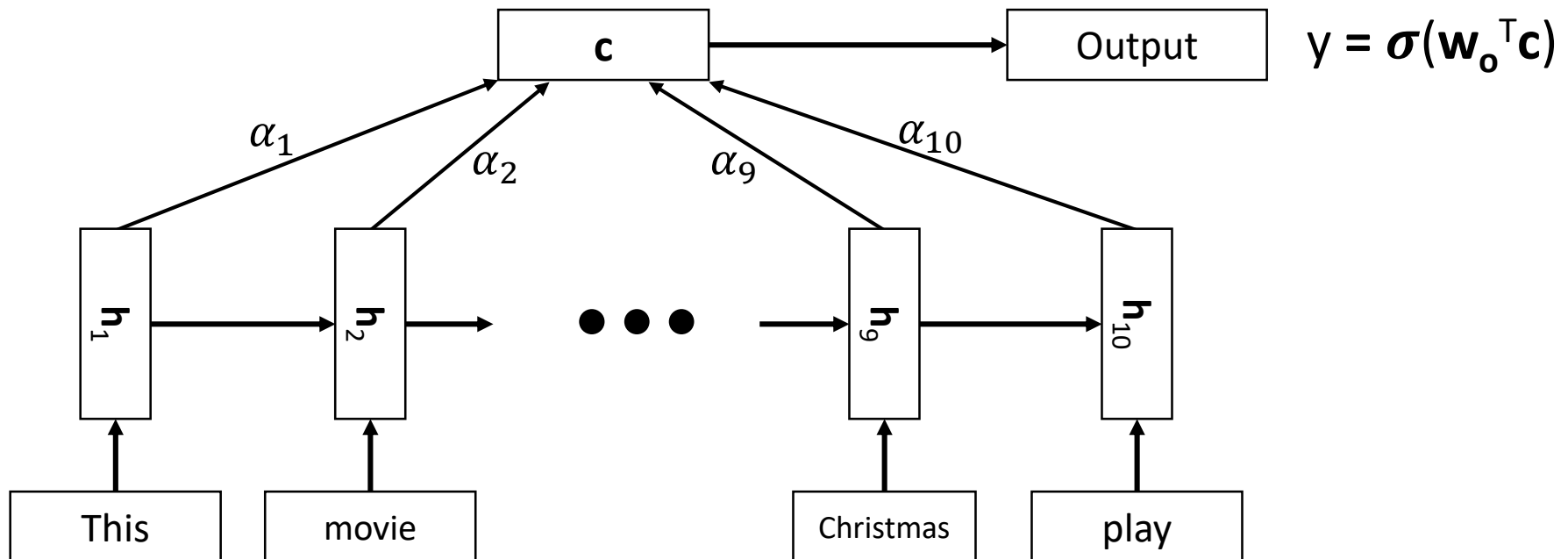
Attention models

- Bahdanau, Cho, Bengio, 2014
 - English-French translation using RNN
- Let's use hidden layers from all timesteps to make predictions



Attention models

- Bahdanau, Cho, Bengio, 2014
 - English-French translation using RNN
- Let's use hidden layers from all timesteps to make predictions

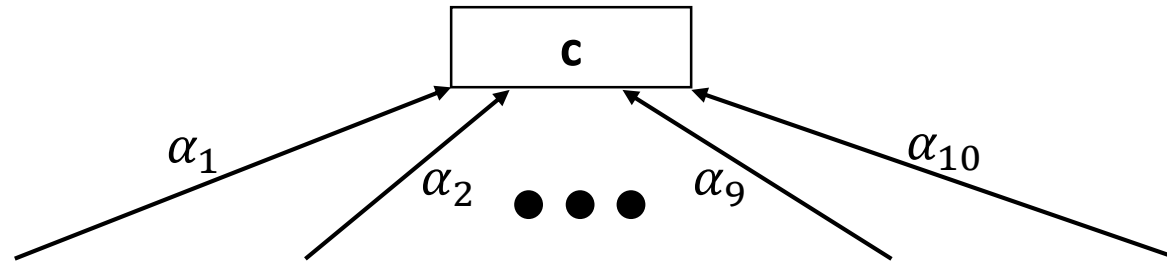


Attention models

- Attention, what is it good for besides improved performance?

Attention models

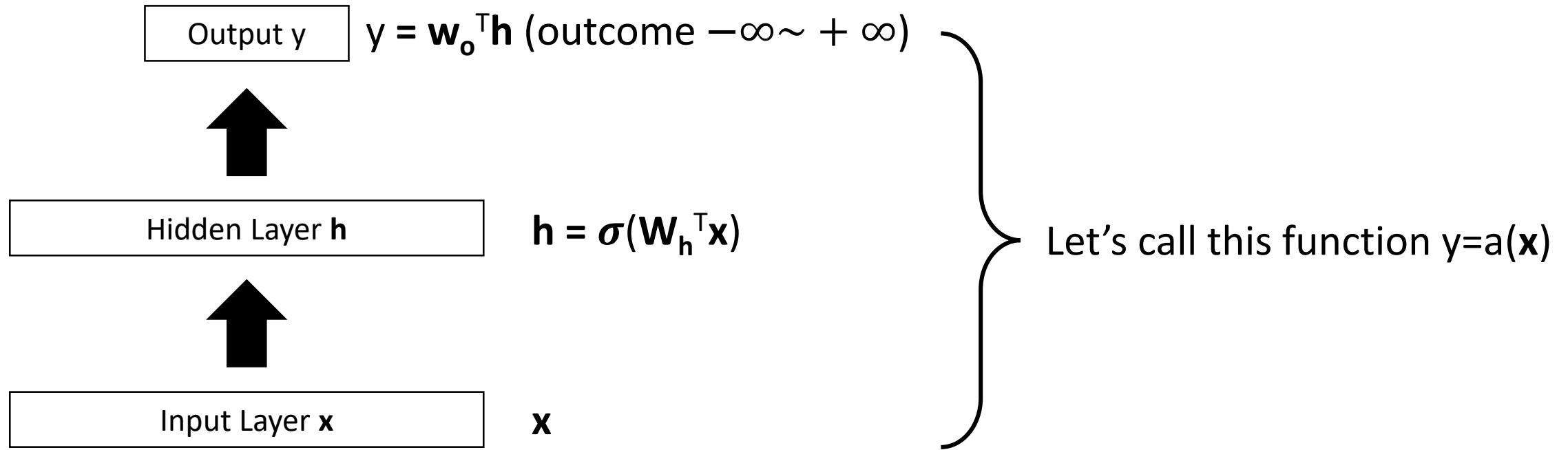
- Attention, what is it good for besides improved performance?



- Now **c** is an explicit combination of all past information
 - $\alpha_1, \alpha_2, \dots, \alpha_{10}$ denote the usefulness from each word
 - We can tell which word was used the most/least to the outcome
- Attentions α_i are generated using an MLP

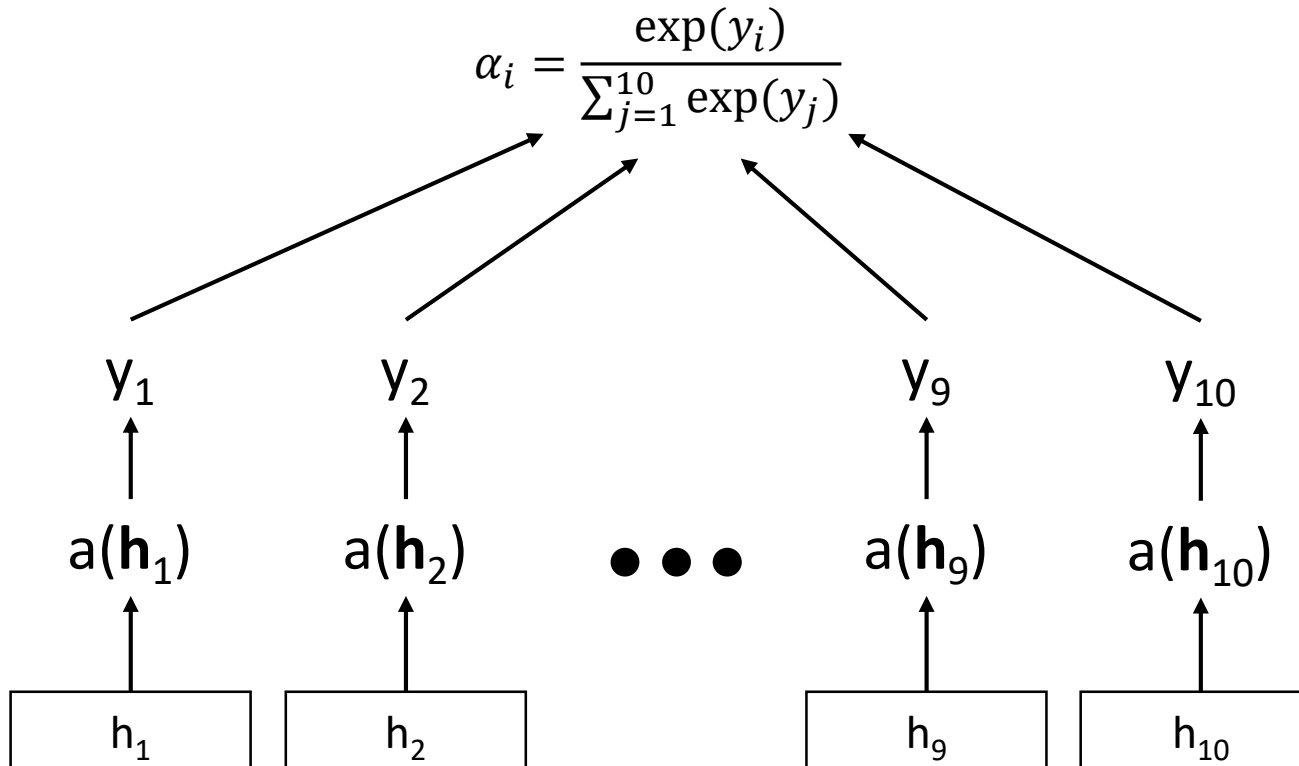
How to generate the attentions α_i ?

- Use another feedforward neural network model



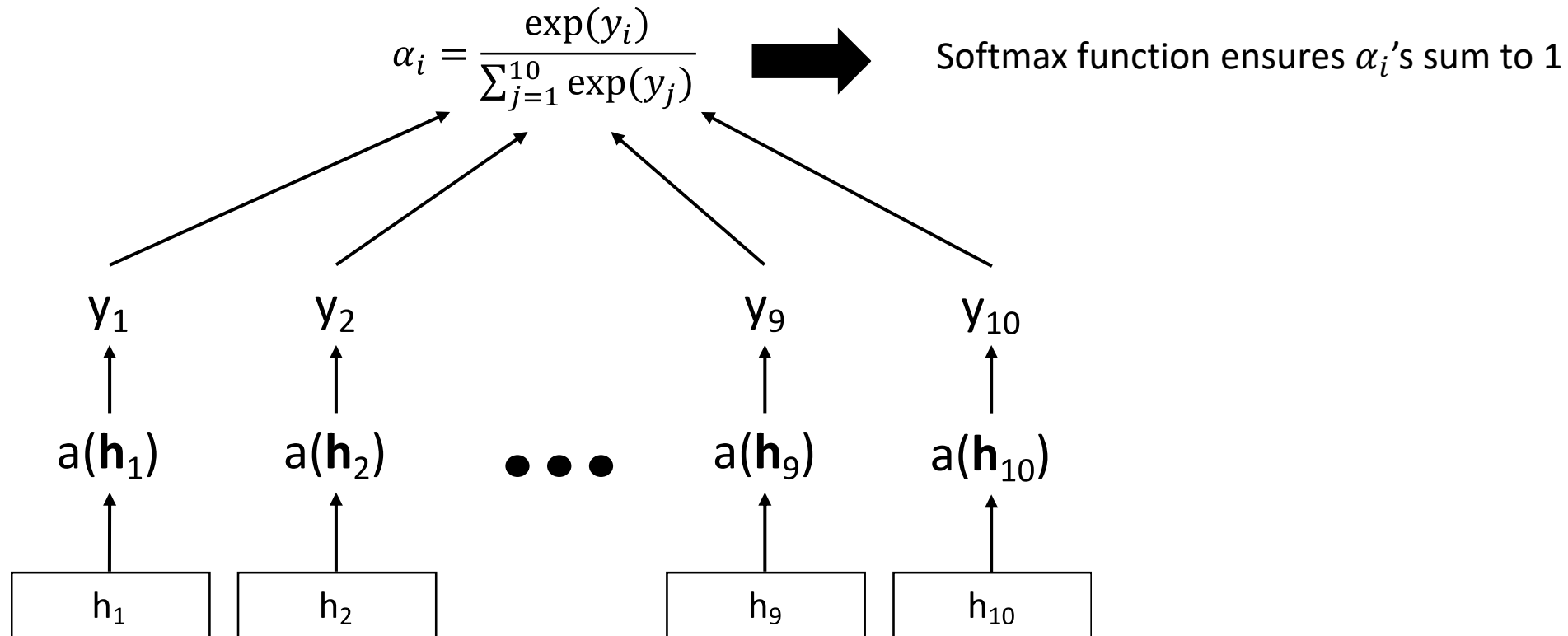
How to generate the attentions α_i ?

- Use function $a(\cdot)$ for each $\mathbf{h}_i$
 - Feed the scores $y_1, y_2, \dots, y_{10}$ into the Softmax function



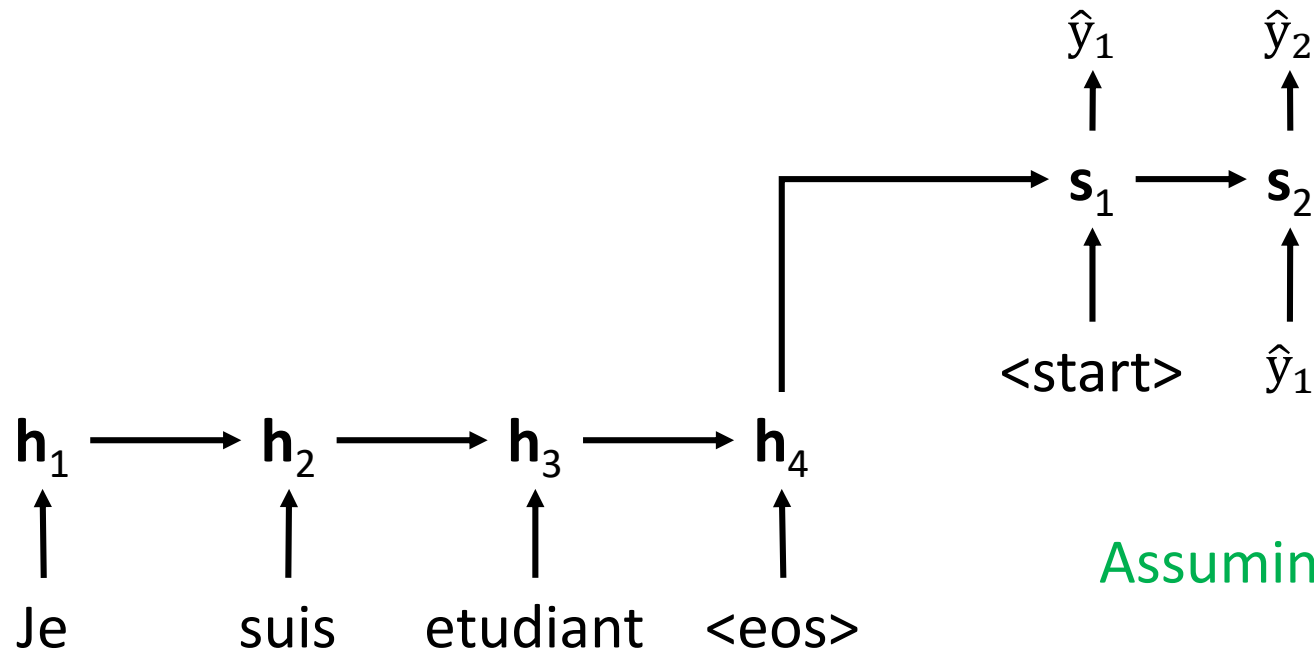
How to generate the attentions α_i ?

- Use function $a(\cdot)$ for each $\mathbf{h}_i$
 - Feed the scores $y_1, y_2, \dots, y_{10}$ into the Softmax function



Attention in Seq2Seq

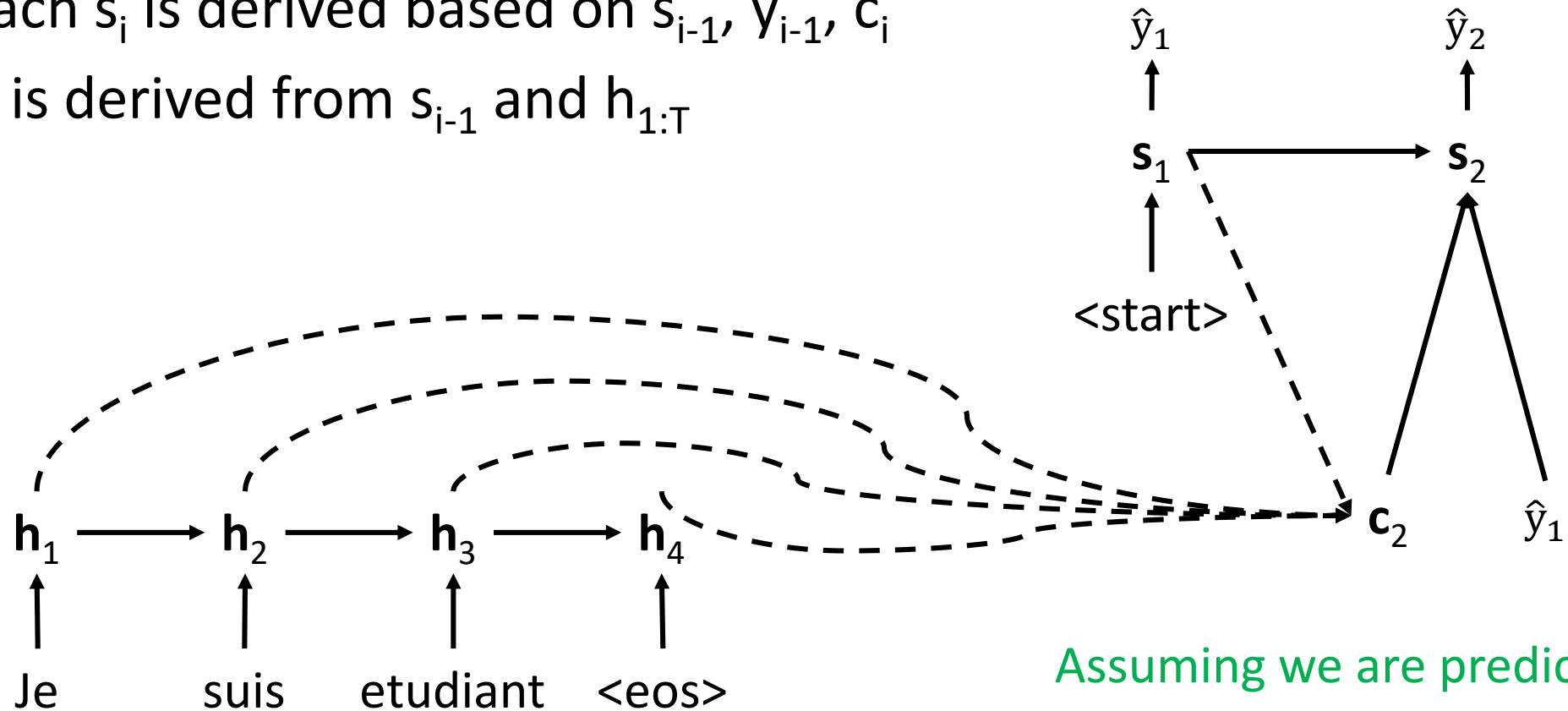
- Each y_i is predicted based on s_i
- Each s_i is derived based on s_{i-1}, y_{i-1}



Assuming we are predicting y_2

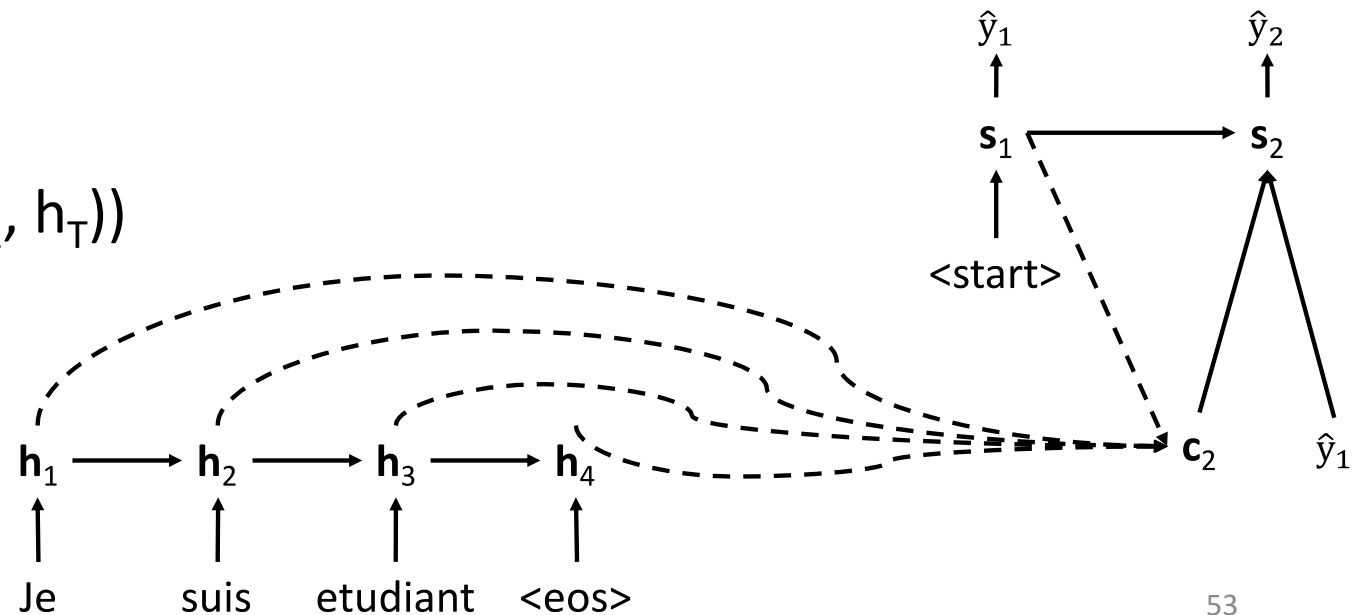
Attention in Seq2Seq

- Each y_i is predicted based on s_i
- Each s_i is derived based on s_{i-1}, y_{i-1}, c_i
- c_i is derived from s_{i-1} and $h_{1:T}$



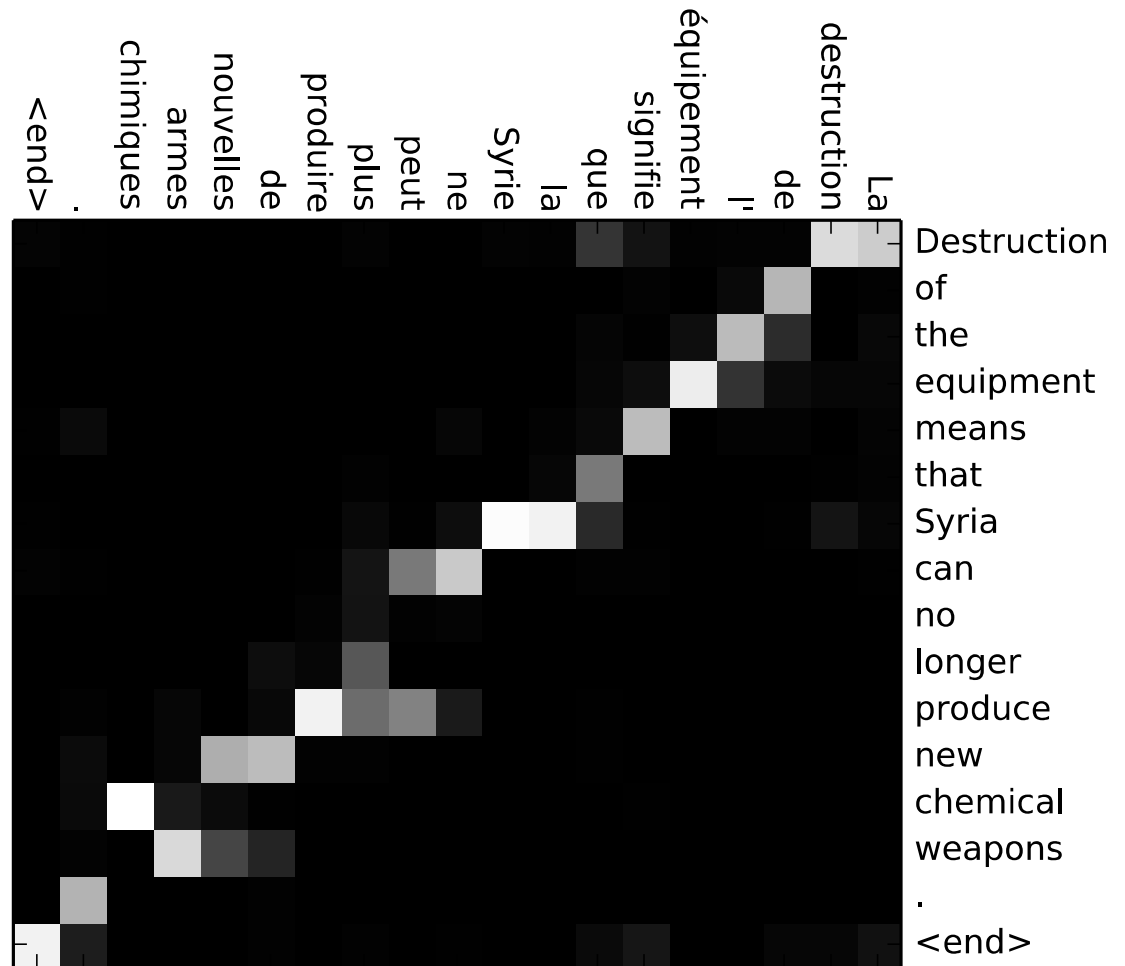
Attention in Seq2Seq

- Each y_i is predicted based on s_i
 - $y_i = \text{Softmax}(W_w s_i + b)$
- Each s_i is derived based on s_{i-1} , y_{i-1} , c_i
 - $s_i = \text{RNN}(s_{i-1}, [y_{i-1}; c_i]_{\text{concat}})$
- c_i is derived from s_{i-1} and $h_{1:T}$
 - $c_i = \text{sum}(\alpha_i * h_i)$
 - $\alpha_i = \text{Softmax}(f(s_{i-1}, h_1), \dots, f(s_{i-1}, h_T))$
 - $f(s_{i-1}, h_j) = s_{i-1}^T W_f h_j$



Attention Example

- English-French translation
 - Bahdanau, Cho, Bengio 2014



AI504: Programming for Artificial Intelligence

Week 10: Recurrent Neural Network

Edward Choi

Grad School of AI

edwardchoi@kaist.ac.kr