Machine Learning 410

Lesson 12

Introduction to Neural Machine Translation with RNNs

Steve Elston

Reminders

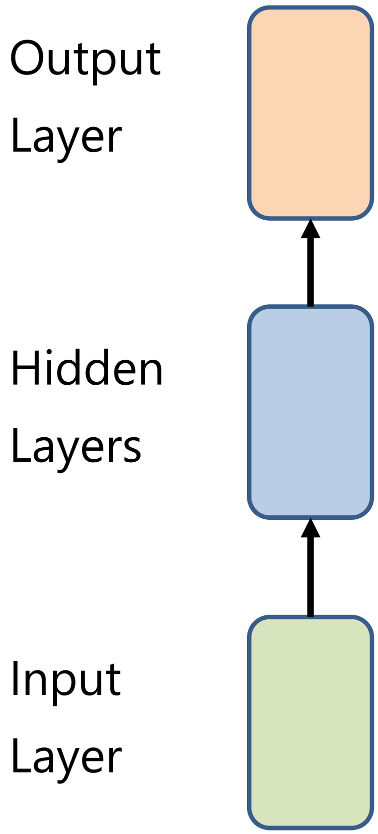
- Discussion – get an easy 5 points!
  - Discussion 8 closes Tonight, Nov 21.
  - Discussion 9 closes Thursday, Dec 5.
- Homework – **Updated homework is in Canvas**
  - Homework 7 due December 2

Outline

- Review of recurrent neural network (RNN) architectures
- The long-short term memory (LSTM) unit
- Bidirectional RNNs (BiRNN)
- Introduction to statistical machine translation (SMT)
- Evaluation of STM models
- Overview of neural machine translation (NMT)
- The attention mechanism for NMT
- Google translate

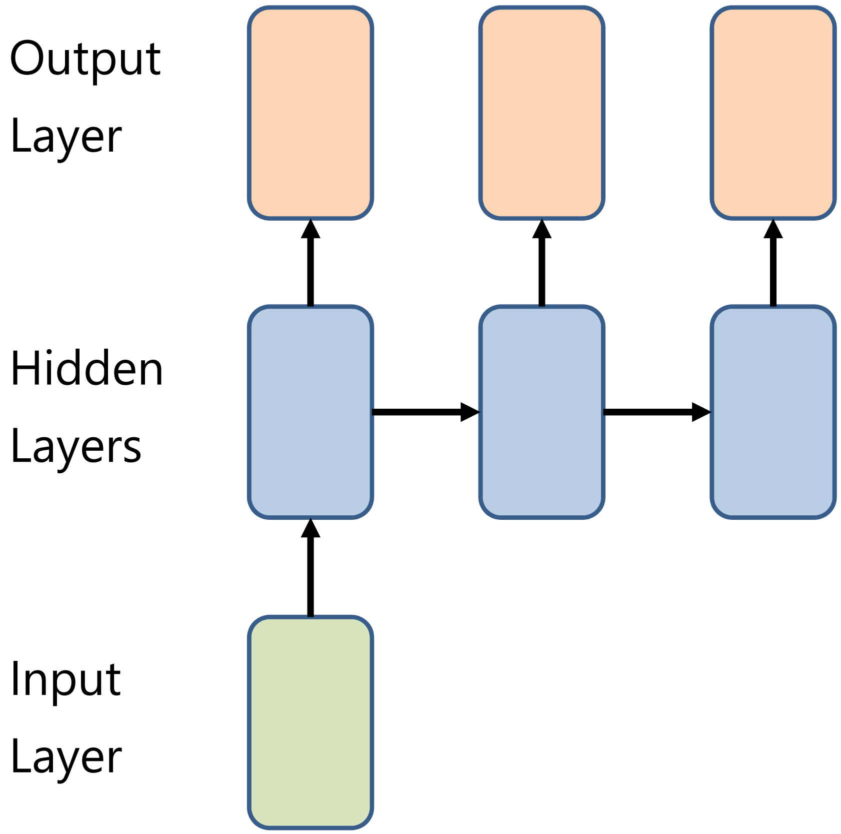
RNN Architectures

Feedforward network



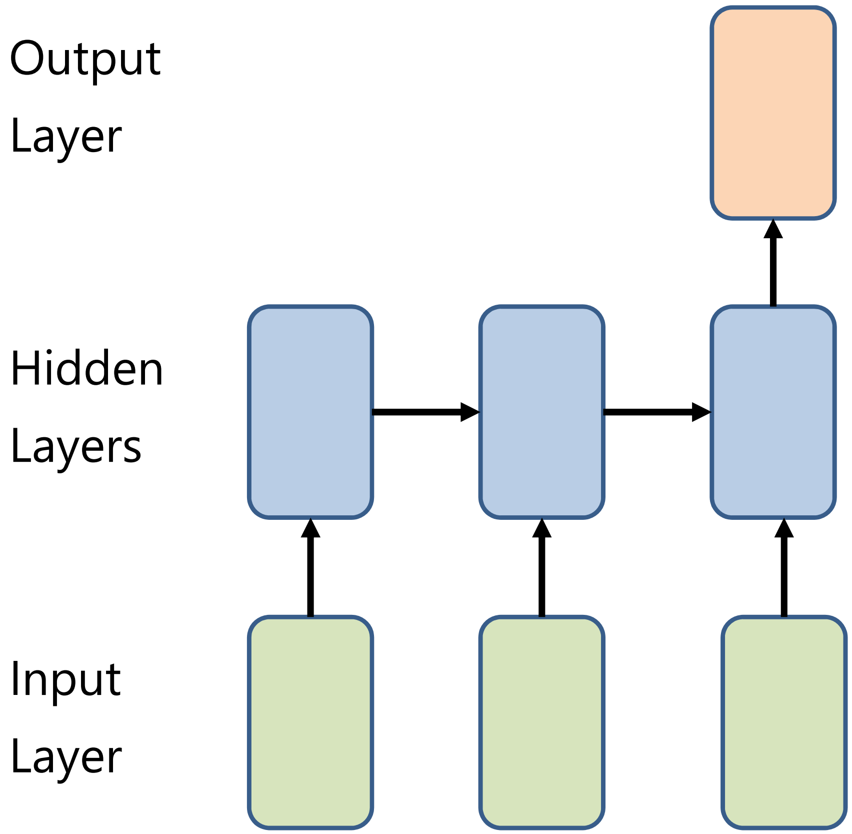
RNN Architectures

Generative model where input vector generates output sequence



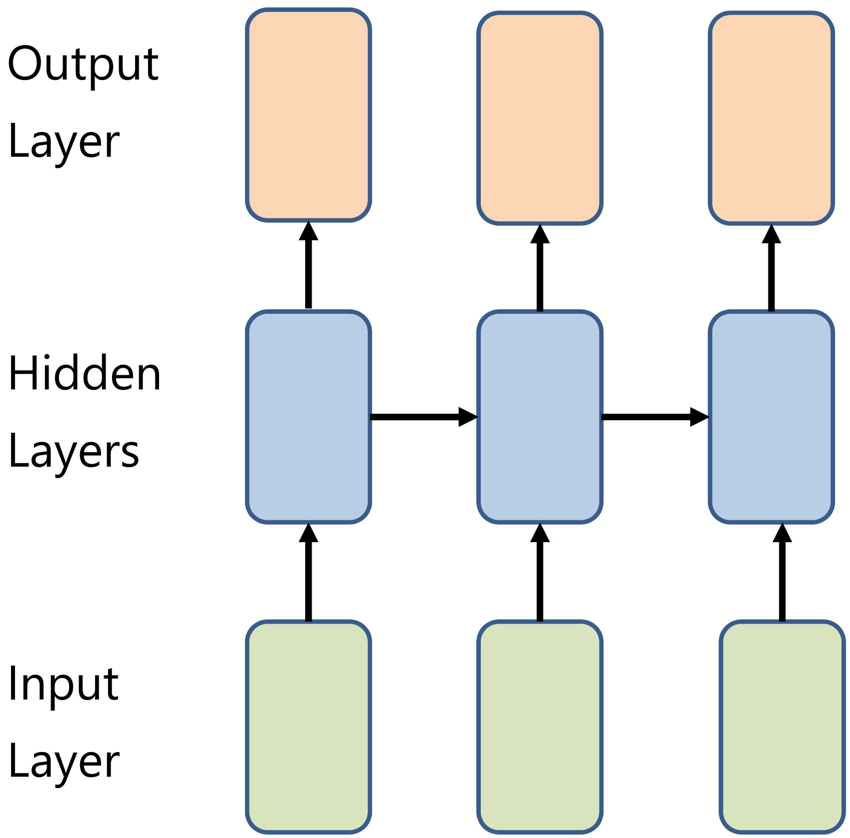
RNN Architectures

Classification of input sequence



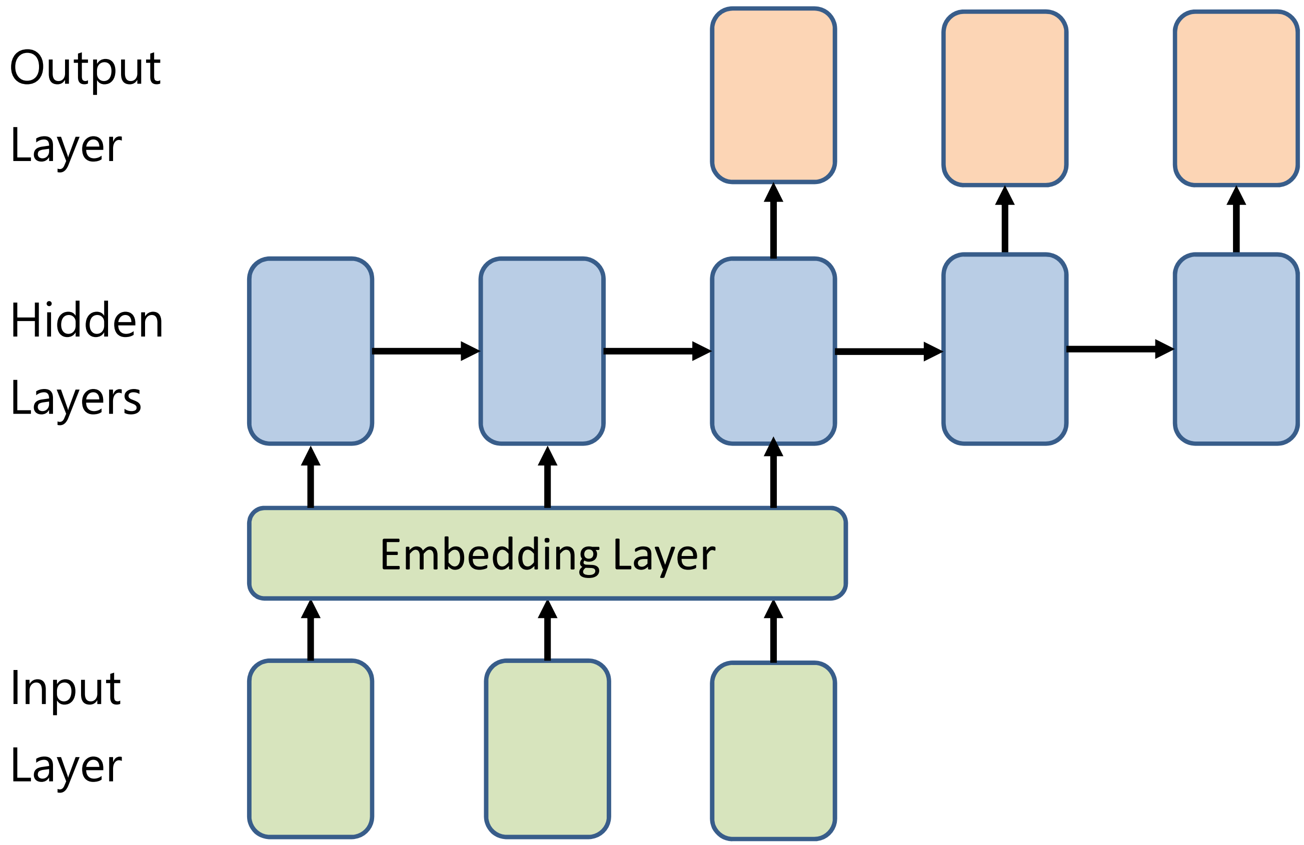
RNN Architectures

Sequence-to-sequence model – Generative



RNN Architectures

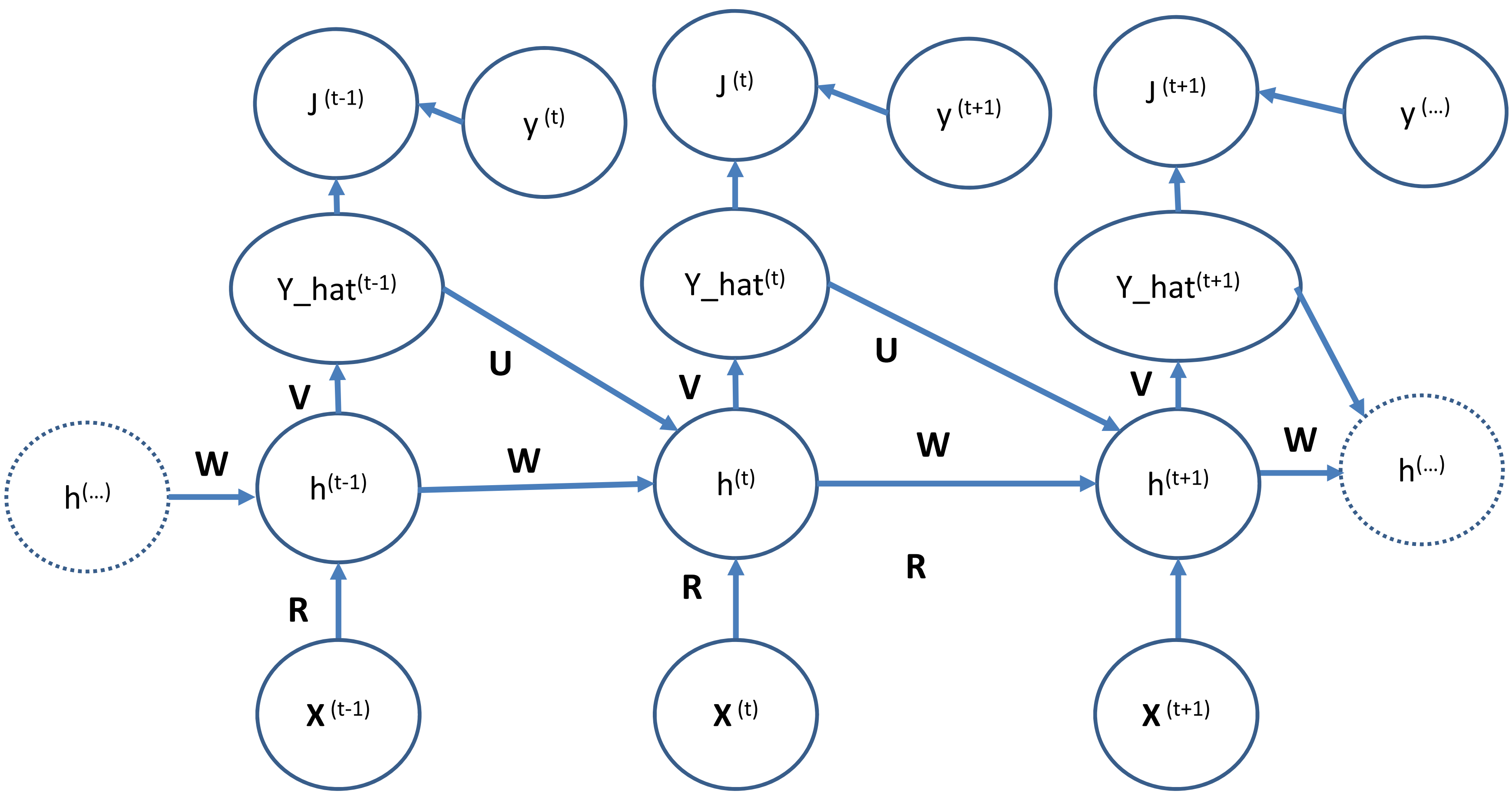
Sequence-to-sequence model with **context** and embedding



Sequence Generation with RNNs

- Given an input value **x**, we want to generate an output sequence **y**.
  - Activation is function of input and last output: $Y_t = h_{t-1}(y_{t-1},x)$
  - Is a generative model
- Train by minimizing the loss with respect to the desired response
  - Loss function = $J(U, V, W)$
- Applications:
  - Response to question – chat bots
  - Caption images
  - Machine translation

Sequence Generation with RNNs



Long-Short Term Memory

- Problems training simple recurrent architectures have led to the development of better approaches
  - Vanishing and exploding gradients common
  - Consider the following recurrent relationship:

$$s^{(t+n)} = f(f(\ldots.f(f(s^{(t)};\theta));\theta)$$

For a long recurrence, large n:

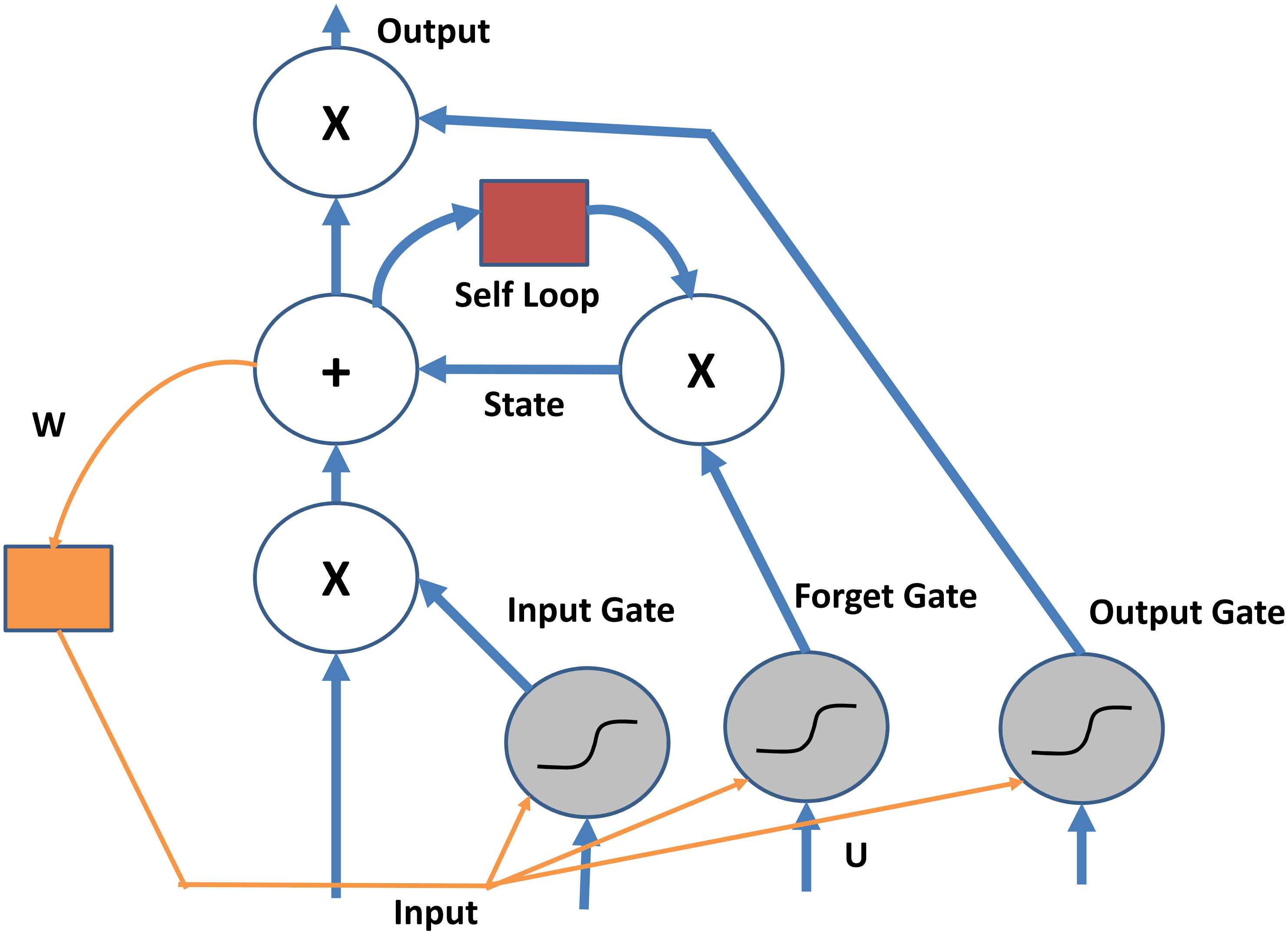
If $f(s;\Theta)$ > 1.0 the gradient grows exponentially

If $f(s;\Theta)$ < 1.0 the gradient vanishes

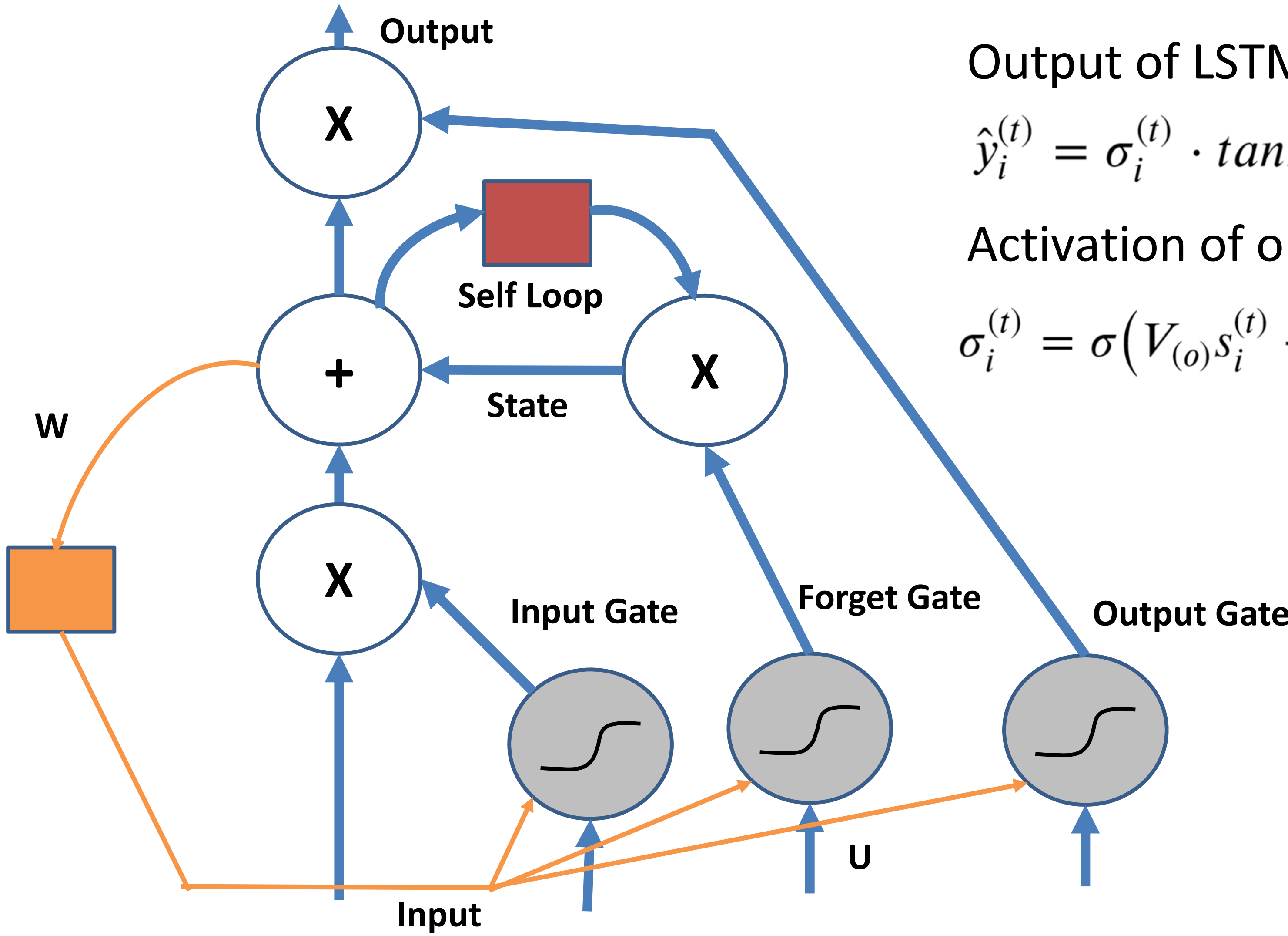
Long-Short Term Memory

- How can one create a neural net for modeling sequences with stable gradients?
  - **Memory** lets the NN operate at multiple time scales
  - **Forget gates** break the recurrence relationship and stabilize the gradient
- The **Long-Short Term Memory (LSTM)** neural network was an early architecture using memory and forget gates
- LSTM used for speech recognition, handwriting generation, machine translation, etc.

Long-Short Term Memory



Long-Short Term Memory



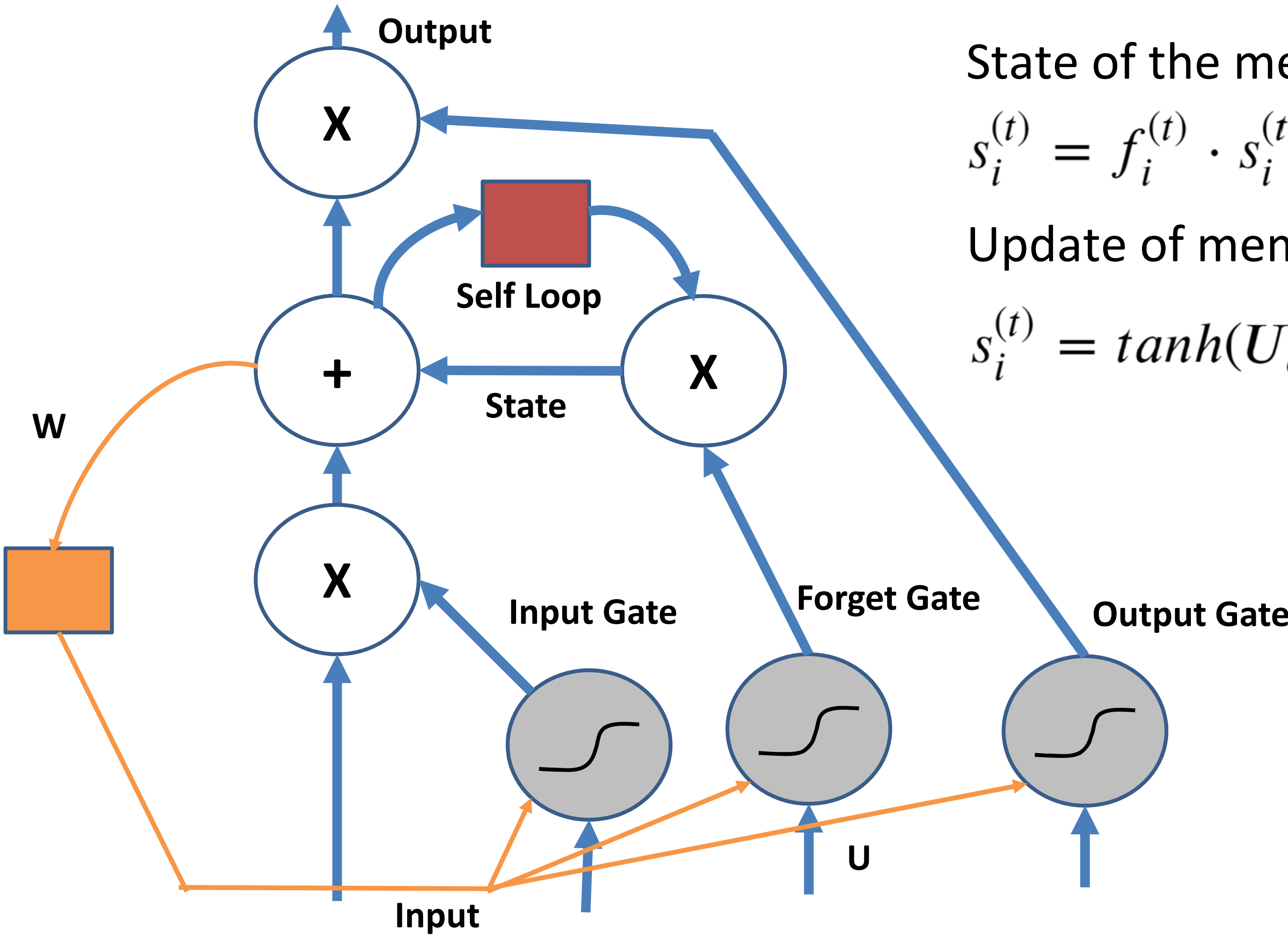
Output of LSTM unit

$$\hat{y}_i^{(t)} = \sigma_i^{(t)} \cdot tanh(y_i^{(t)})$$

Activation of output gate

$$\sigma_i^{(t)} = \sigma\left(V_{(o)} s_i^{(t)} + U_{(o)} x_i^{(t)} + W_{(o)} h_i^{(t-1)}\right)$$

Long-Short Term Memory



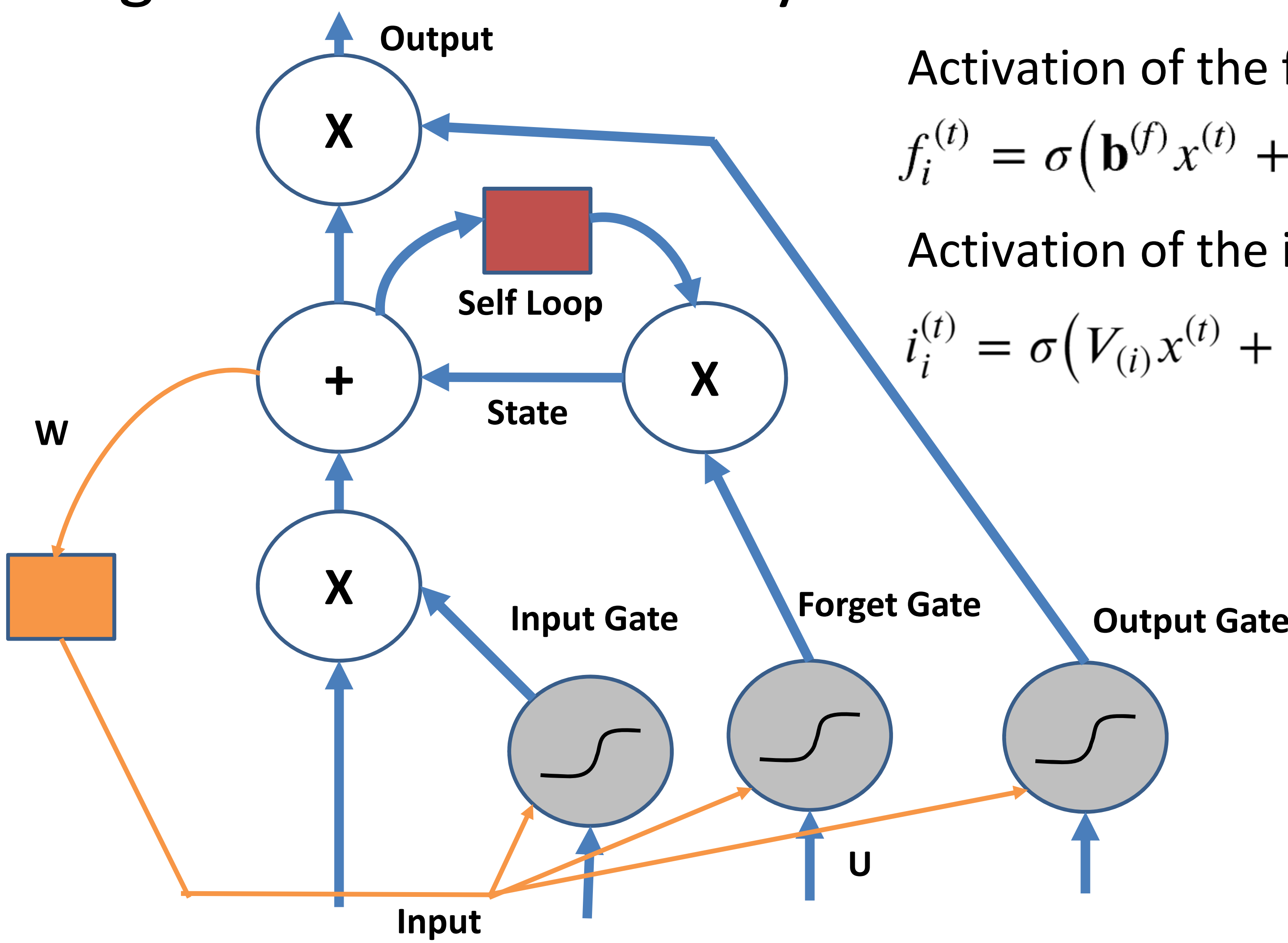
State of the memory loop

$$s_i^{(t)} = f_i^{(t)} \cdot s_i^{(t)} + i_i^{(t)} \cdot s_i^{(t-1)}$$

Update of memory state

$$s_i^{(t)} = tanh(U_c x_i^{(t)} + W_c h_i^{(t-1)})$$

Long-Short Term Memory



Activation of the forget gate

$$f_i^{(t)} = \sigma\left(\mathbf{b}^{(f)} x^{(t)} + \mathbf{U}^{(f)} x^{(t)} + \mathbf{W}^{(f)} h^{(t-1)}\right)$$

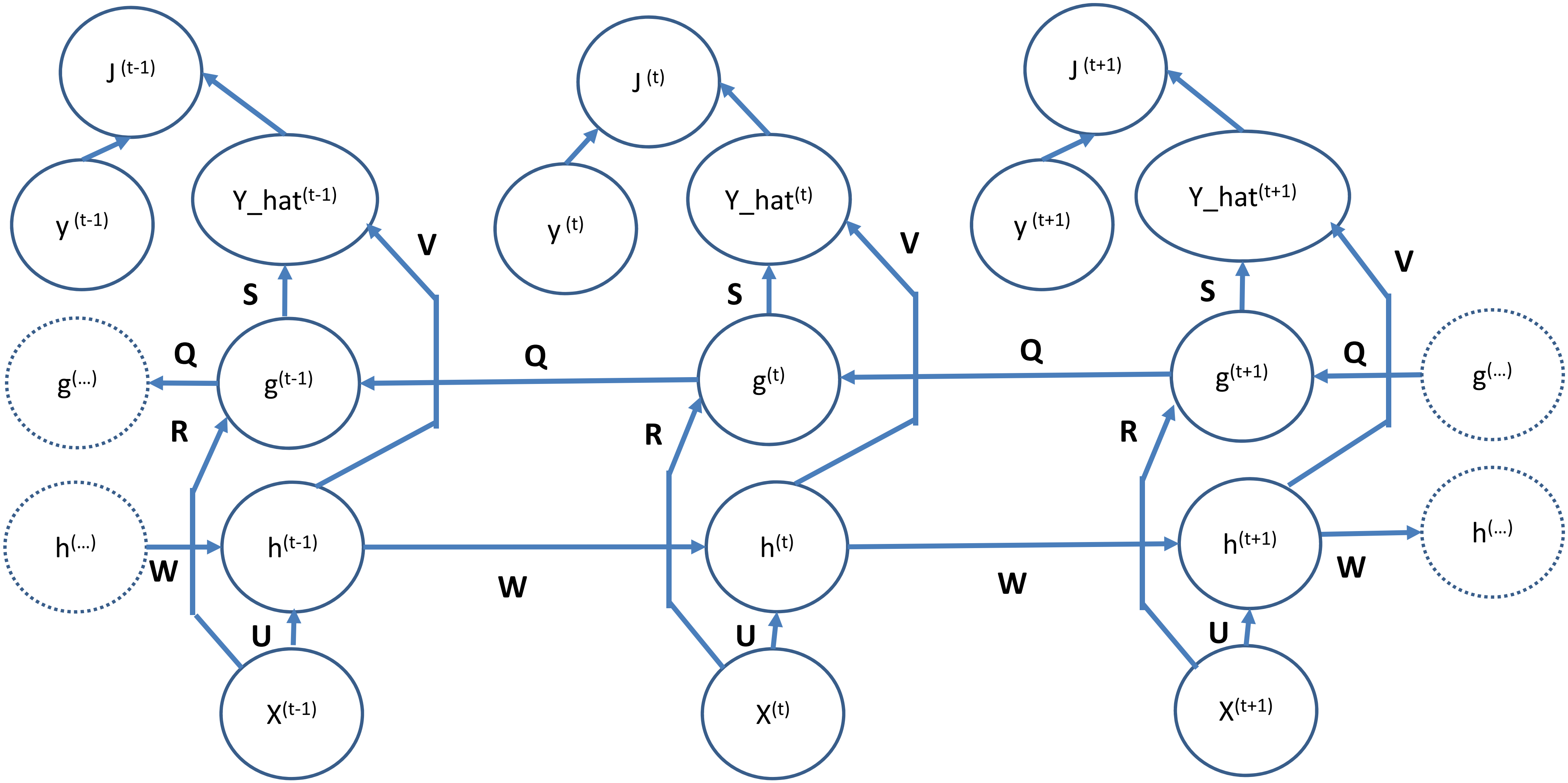
Activation of the input gate

$$i_i^{(t)} = \sigma\left(V_{(i)} x^{(t)} + U_i x_(i)^{(t)} + W_{(i)} h_i^{(t-1)}\right)$$

Bidirectional RNNs

- Not all sequential relationships are causal
  - A natural language phrase can be parsed in both directions
  - Hand writing recognition can proceed from either end
  - Figure captioning has no preferred direction
- For non-causal sequences we can use **bidirectional RNNs**
- Bidirectional RNNs trained using BPTT in both directions

Bidirectional RNNs



Statistical Machine Translation

How to build a statistical model to translate from one language to another?

- Statistical machine translation has a long history, starting in the 1950s
- Goal:
  - Given an **input sequence**, $\mathbf{x} = x_1, x_2, \ldots, x_n$, of tokens in a first language
  - Find the most probable or **target output sequence**, $\mathbf{y} = y_1, y_2, \ldots, y_m$, in a second lanuage

Statistical Machine Translation

How to build a statistical model to translate from one language to another?

- The most probability of the output sequence ***y***, given the input sequence ***x*** follows this relationship

$$p(\mathbf{y}|\mathbf{x}) \propto p(\mathbf{x}|\mathbf{y})p(\mathbf{y})$$

- Where,

  $p(\mathbf{x}|\mathbf{y})$ is the **translation model**, or likelihood of sequence ***x*** given sequence ***y***

  $p(\mathbf{y})$ is the language model, or probability of sequence ***y*** in the target lanuage

Statistical Machine Translation

How to build a statistical model to translate from one language to another?

- Find most probably tartget sequence

$$p(\mathbf{y}|\mathbf{x}) \propto p(\mathbf{x}|\mathbf{y})p(\mathbf{y})$$

- Can find weights for the maximum likelihood expression
  $\mathbf{w} = w_1, w_2, \ldots, w_N$

$$logp(\mathbf{y}|\mathbf{x}) = \sum_{n=1}^{N} w_n p(\mathbf{y}|\mathbf{x}) + logZ(\mathbf{x})$$

- Where $Z(\mathbf{x})$ is the normalization constant not dependent on ***y***

Evaluation of SMT Models

The BLEU score

- The **Bilingual Translation Understudy** or **BLEU score** is used to compare SMT models.
- BLEU is a comparison between results of an SMT model and results of expert human translators
- The BLEU compares a hypothesis of *i-gram* tuples, *H(i)*
- The BLEU is the average proportion of *i-gram* matches
- A model with a higher BLEU is considered superior

Evaluation of SMT Models

The BLEU score

The **BLEU score** for a sequence length N is computed:

$$BLEU = \{\prod_{i=1}^{N} P(i)\}^{\frac{1}{N}}$$

Where,

$$P(i) = \frac{Matched(i)}{H(i)}$$

Evaluation of SMT Models

The BLEU score

- *H(i)* is the number of *i-gram* tuples in each hypothesis

  For hypothesis of length *n*, examples are: *H(1) = n*, *H(2) = n-1*, *H(3) = n-2*

- And the number of matches:

$$Matched(i) = \sum_{t_i} min\{C_h(t_i), \max_j C_{hj}(t_i)\}$$

  Where, for i-gram tuple t_i

  $C_h(t_i)$ = number of times t_i occurs in the hypothesis to be tested

  $C_{hj}(t_i)$ = number of times t_i occurs in the j-th reference

Evaluation of SMT Models

The BLEU score

- The **BLEU score** is dependent on the sequence length
- A commonly used adjustment is the **brevity penalty,** ρ
- The adjusted BLEU score is then:

$$BLEU_\rho = exp\Big(min(0, \frac{n - L}{n})\Big)\{\prod_{i=1}^{N} P(i)\}^{\frac{1}{N}}$$

Where:

n = length of the hypothesis

L = length of the reference sequence

Evaluation of SMT Models

BLEU score example

- Start with French sentence:

  La voiture est dans l'allée.

- An expert English translation is:

  The car is in the drive.

Evaluation of SMT Models

BLEU score example

- Start with the expert English translation:

  The car is in the drive.

- There are 6 -1 = 5, 2-grams for this sentence:

Reference 2-grams
The car
Car is
Is in
In the
The drive

A perfect MT is identical so:

Matched = 5

H(2) = 5

$\rho = 1.0$

BLEU = 100.0%

Evaluation of SMT Models

BLEU score example

- A possible MT is:

  The drive contains the car.

- There are 5 -1 = 4, 2-grams for this sentence:

Reference 2-grams	MT 2-grams	Match
The car	The drive	1
Car is	Drive contains	0
Is in	Contains the	0
In the	The car	1
The drive		

Matched = 2

H(2) = 5

n = 6

l = 5

ρ = 1.18

BLEU = 47.3%

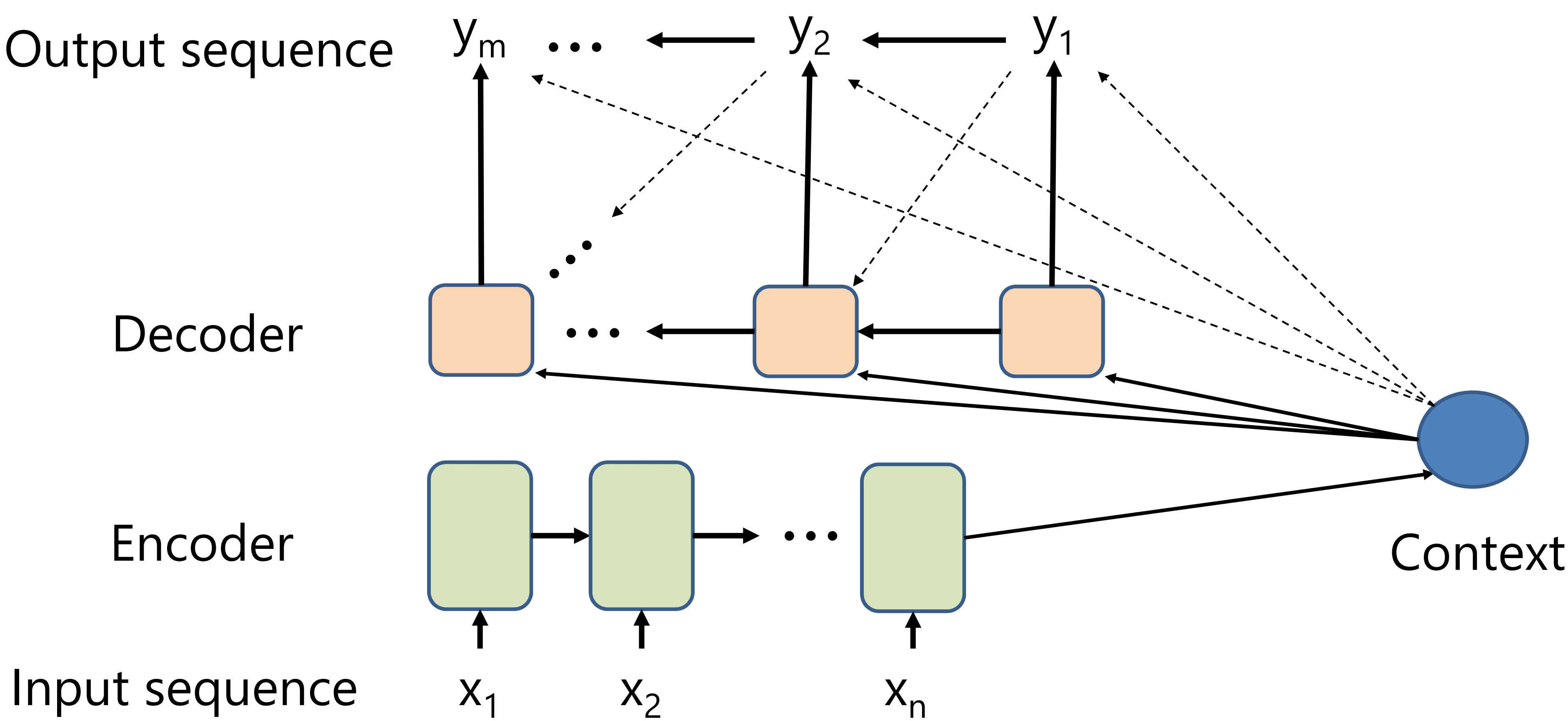
Neural Machine Translation

Use neural networks as function approximators for machine translators

- Architecture has three components:
  - **Encoder** for the input sequence
  - **Context vector** – a hidden state
  - **Decoder** generates output sequence
- Finds maximum likelihood target sequence

NMT Architectures

Encoder-decoder model with context vector



Neural Machine Translation

Use neural networks as function approximators for machine translators

- The hidden state updates as a function current hidden state and input

$$h_t = f(h_{t-1}, x_t)$$

Where, *f()* is the activation function of the LSTM

- The encoder creates a sequence of hidden states given the input sequence

$$h_1, h_2, \ldots, h_n = Encoder_{RNN}(x_1, x_2, \ldots, x_n)$$

Neural Machine Translation

Use neural networks as function approximators for machine translators

- The first layers of a NMT encoder network are a bidirectional RNN (BiRNN)
- The hidden states of the forward layer are:

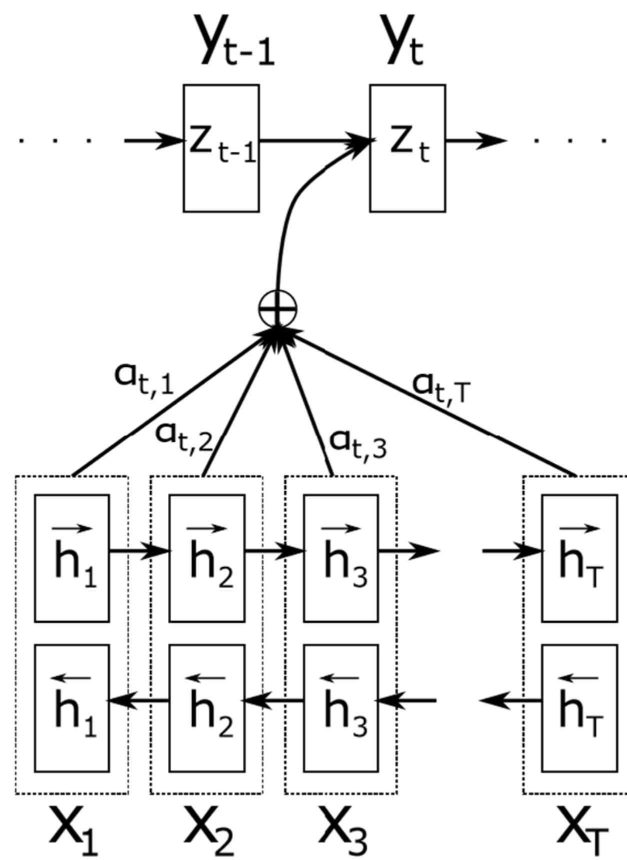
$$\{\overrightarrow{h_1}, \overrightarrow{h_2}, \ldots, \overrightarrow{h_T}\}$$

- The hidden states of the reverse layer are:

$$\{\overleftarrow{h_T}, \overleftarrow{h_{T-1}}, \ldots, \overleftarrow{h_1}\}$$

- And, the context is:

$$c_t = [\overrightarrow{h_t}; \overleftarrow{h_t}]$$



Neural Machine Translation

Use neural networks as function approximators for machine translators

- Given weights, $\mathbf{w} = w_1, w_2, \ldots, w_N$, the encoder uses the context and its hidden state

$$p(y_1, y_2, \ldots, y_m) = \prod_{t=1}^{m} p(y_t | \{y, c, h\}_{\leq m})$$

- Taking logs of both sides, gives:

$$logp(y_1, y_2, \ldots, y_m) = \sum_{t=1}^{m} logp(y_t | \{y, c, h\}_{\leq m})$$

Attention Mechanism

- **Context** is the input to decoder
- A **fixed context vector** has limited representation
  - May compress information required for sequence generation
  - Translation accuracy decreases with sequence length
- Need a better representation!
- Use a **context set**: $c = \{c_1, c_2, \ldots, c_M\}$
- The **weights** of the context give **attention** to the correct sequence in the decoder

Attention Mechanism

How to compute the **attention weights**?

- Compute **attention score** for context c_i as a function of pervious hidden state, context and previous attention weights:

$$e_i^t = f_{ATT}(h_{t-1}, c_i, \{\alpha_j^{t-1}\}_{j=1}^M)$$

- The attention weights are updated:

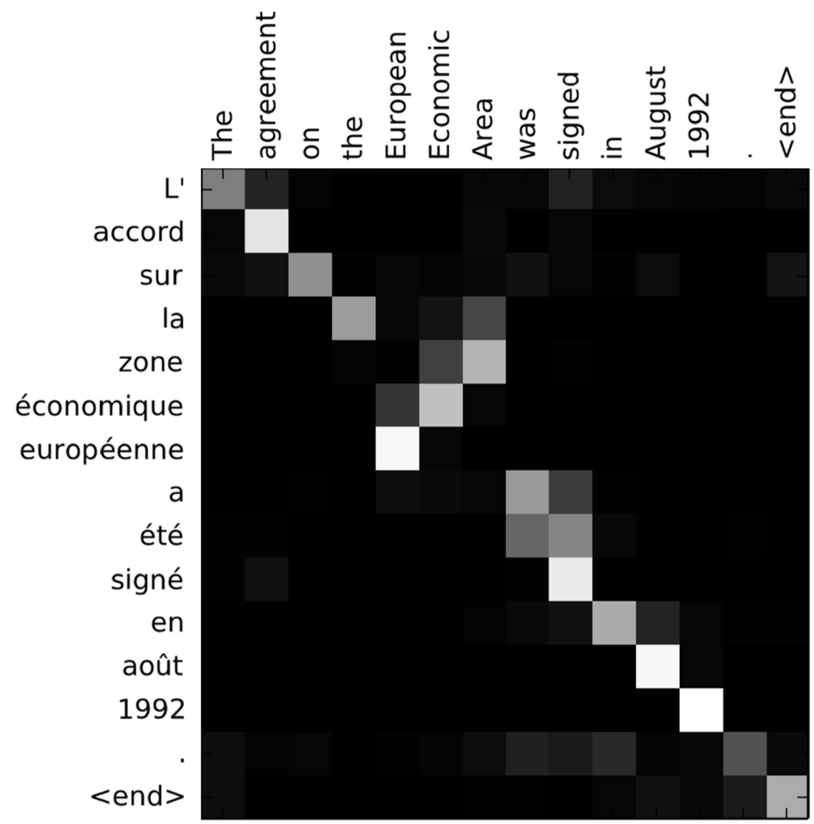
$$\alpha_i^t = \frac{exp(e_i^t)}{\sum_{j=1}^M exp(e_j^t)}$$

- The attention weights are the **probability that decoder should attend to the context c_i**

Attention Mechanism

Example of attention weights

- Attention weights can be readily visualized
- For the input sequence
- And, the output sequence
- **Probabilities of attending the output sequence given the input sequence**



[Cho, et. al, Describing Multimedia Content using Attention-based Encoder–Decoder Networks, 2015](#)

Attention Mechanism

How to compute the **attention weights**?

- The scores used to update the context weight are a function of the current context set and the attention weights:

$$e_i^t = \phi(\{c_i^t\}_{i=1}^M, \{\alpha_i^t\}_{i=1}^M)$$

- One possibility is to use linear combination of weighted context:

$$e_i^t = \phi(\{c_i^t\}_{i=1}^M, \{\alpha_i^t\}_{i=1}^M) = \sum_{i=1}^M \alpha_i c_i$$

Attention Mechanism

How to compute the **attention weights**?

- Use a neural network as a function approximator
- The neural network has parameters, Θ, which maximize the log-likelihood

$$\mathcal{L}\left(D, \Theta\right) = \frac{1}{N} \sum_{n=1}^{N} log\ p(y^n | x^n, \Theta)$$

- Where the data are the N input and output sequences of the training data; $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$
- For linear model, can compute all the derivatives and use backpropagation to train neural network with stochastic gradient decent

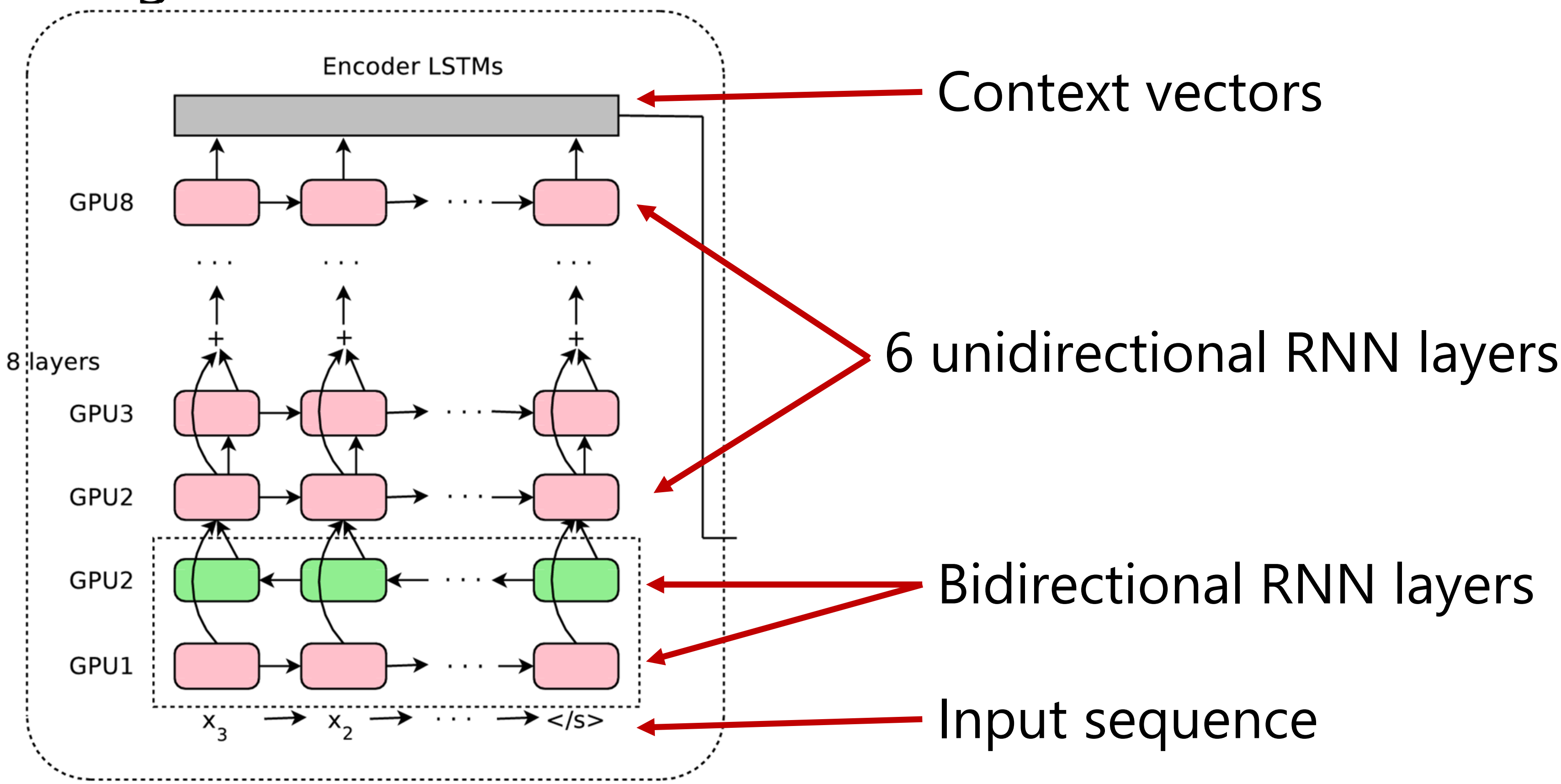
Beam Search in Decoder

Use the beam search method to find the most probable output sequence

- Given the attention probabilities, α_i^t, how can the decoder find the most probable output sequence?
- Use a search heuristic known as **beam search**
- Beam search is a classic AI method originating in the 1970s
- Beam search is a **breath first search method**
  - Search along k highest probability paths
  - Uses less memory than full breath search
- Beam search used to find the sequence with the highest probability or likelihood

Google Translate

Google Translate architecture: Encoder

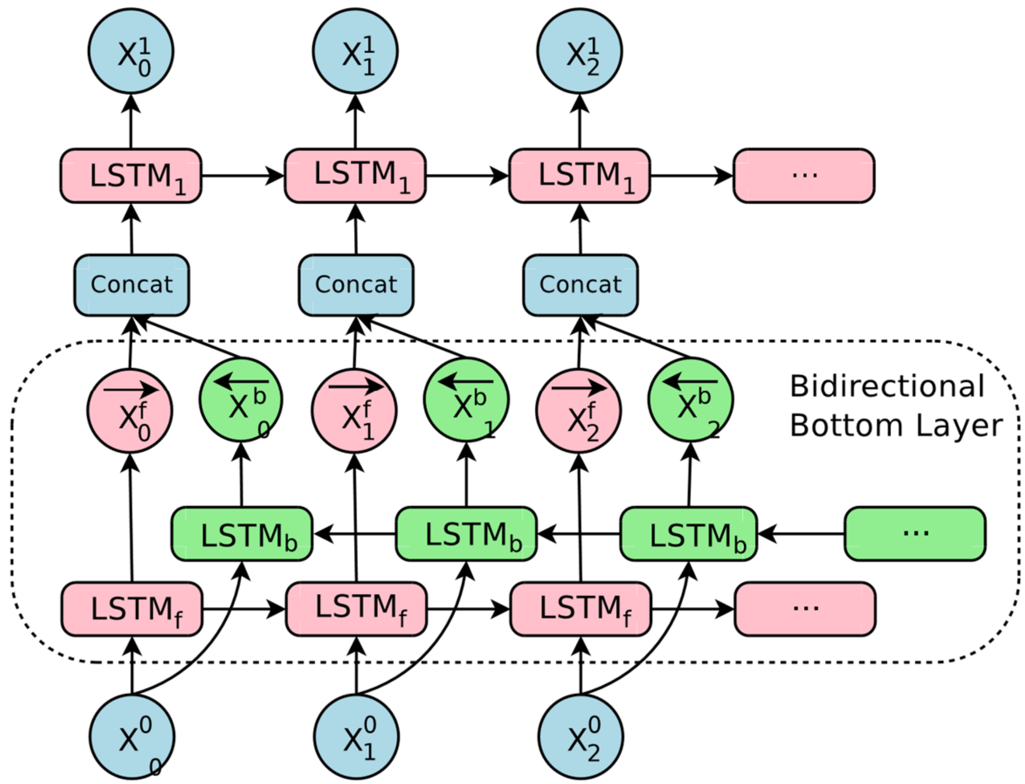


[Wu, et. al, Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation, 2016]

Google Translate

Google Translate architecture

Details of the biRNN layer



[Wu, et. al, Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation, 2016]

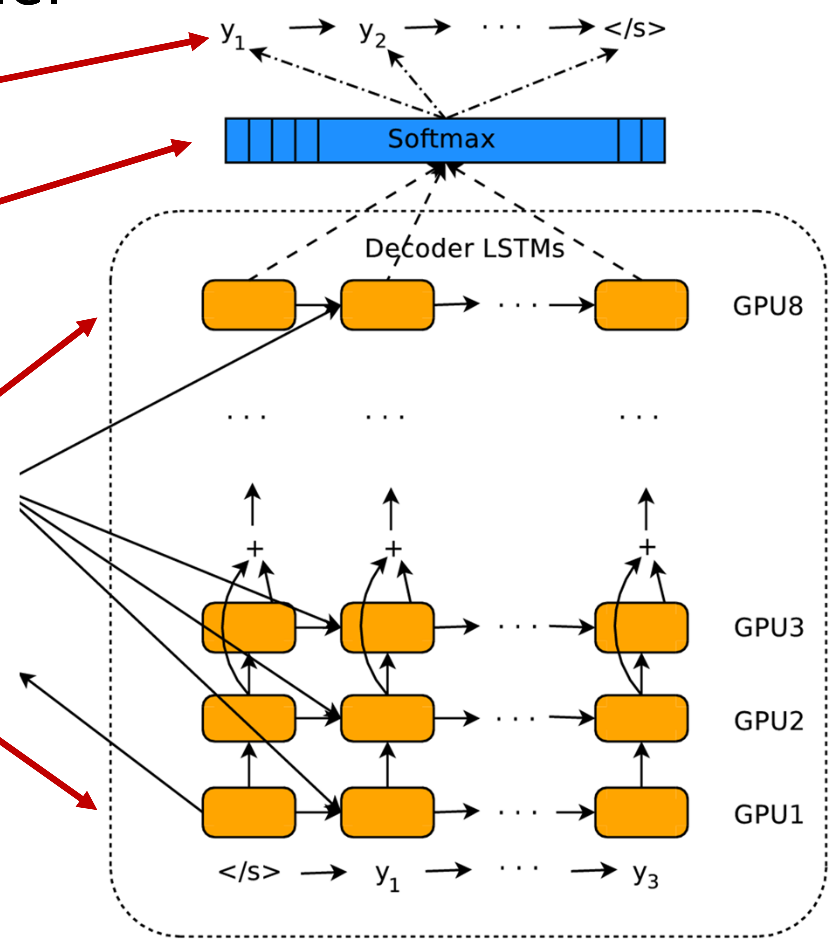
Google Translate

Google Translate architecture: Decoder

Output sequence

Softmax layer

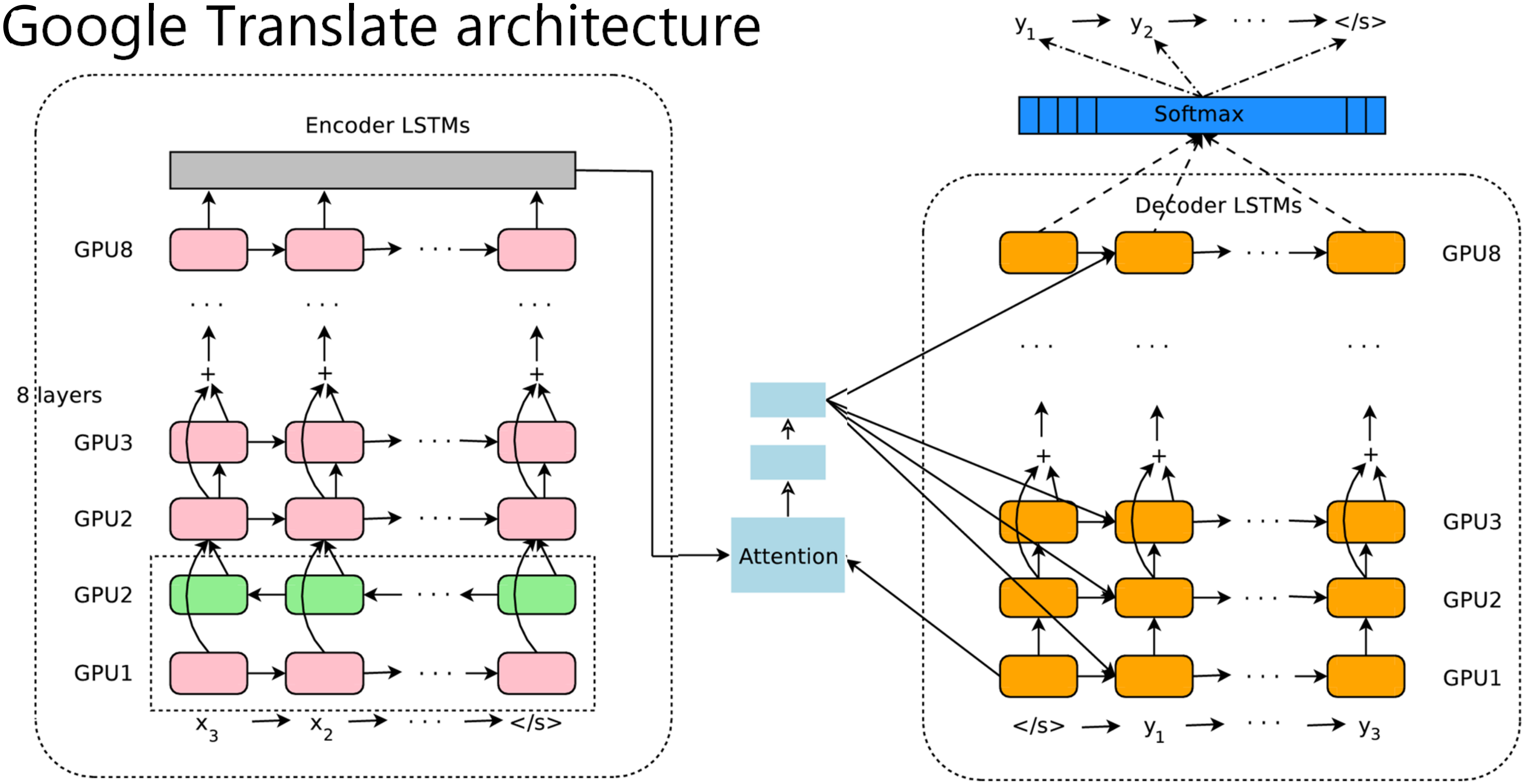
8 RNN layers



[Wu, et. al, Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation, 2016](#)

Google Translate

Google Translate architecture



[Wu, et. al, Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation, 2016]

Google Translate

Beam search for the attention mechanism

- Need a decoder which does not favor particular output sequence
- Standard likelihood score needs adjustments
  - Do not want to favor short sequences resulting from the summation of negative log likelihoods
  - Need to ensure that the entire input sequence is reflected in the output sequence

Google Translate

Beam search for the attention mechanism

The modified likelihood score becomes:

$$\mathcal{L}\Big(D, \Theta\Big) = \frac{1}{lp(Y)} \frac{1}{N} \sum_{n=1}^{N} log\ p(y^n | x^n, \Theta) + cp(D)$$

$$lp(Y) = \frac{(5 + |Y|)^\gamma}{(5 + 1)^\gamma}$$

$$cp(D) = \beta \sum_{i=1}^{|X|} log\Big(min(\sum_{j=1}^{|Y|} \alpha_{i,j}, 1.0)\Big)$$

Where

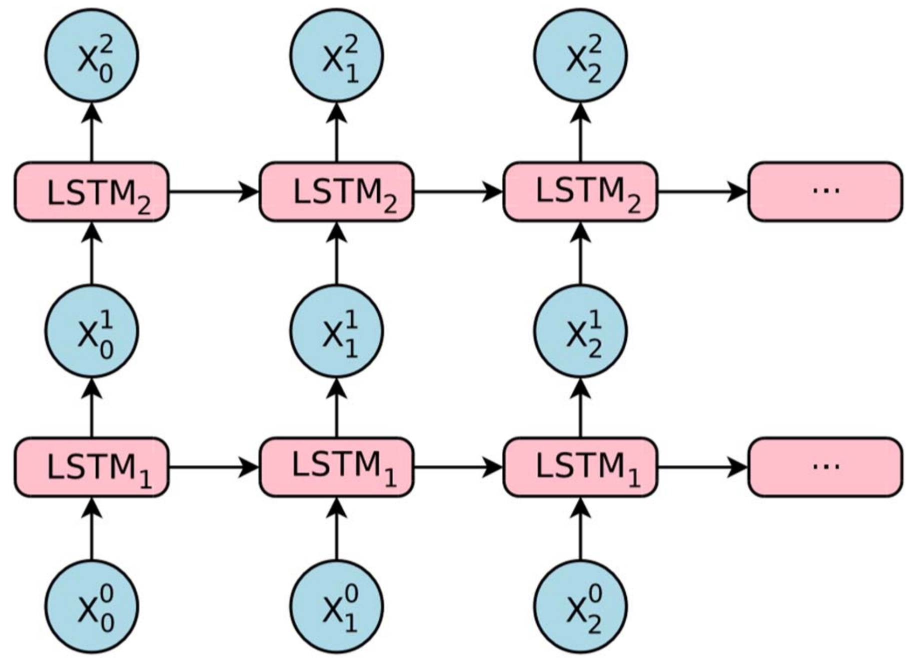
γ = strength of the normalization

β = extent to which translations that fully cover the source sentence are favored

Google Translate

How to train deep LSTM RNNs?

- Stacking many LSTM layers is problematic
- Gradient vanishes
- Solution: use residual connections!



[Wu, et. al, Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation, 2016]

Google Translate

How to train deep LSTM RNNs?

- The following relations govern the update between the i-th and i+1-th standard LSTM layers:

$$h_t^i, m_t^i = LSTM_i(h_{t-1}^i, m_{t-1}^i, x_t^{i-1}, \mathbf{W}^i)$$

$$x_t^i = h_t^i$$

$$h_t^{i+1}, m_t^{i+1} = LSTM_{i+1}(h_{t-1}^{i+1}, m_{t-1}^{i+1}, x_t^i, \mathbf{W}^{i+1})$$

Where, for the i-th layer:

m_i = the LSTM memory

h_i = hidden state

x_i = input

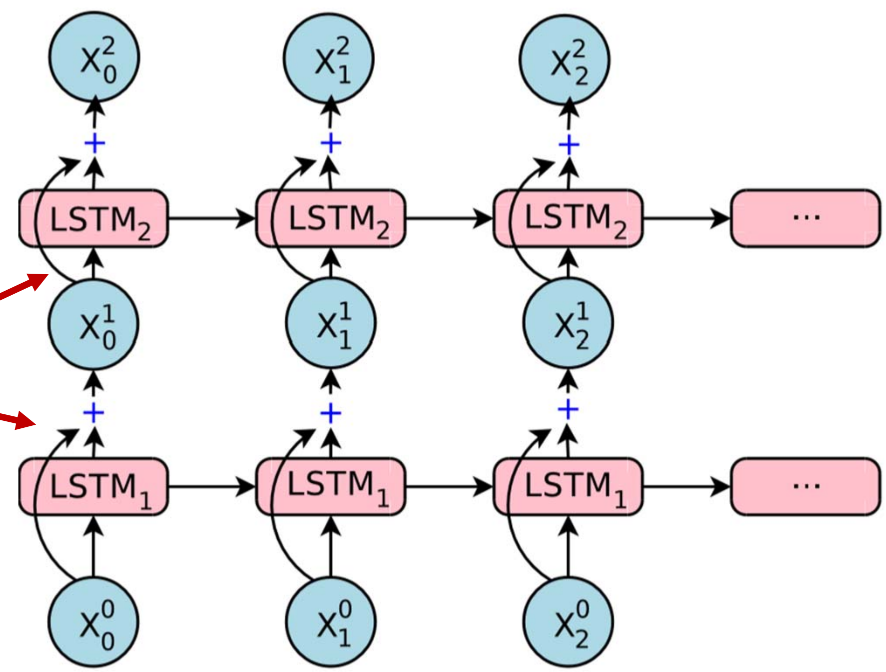
$\boldsymbol{W}_i$ = weight tensor

RNN with residual connections

Google Translate architecture

- How to deal with vanishing gradient in deep RNN?
- Use residual connections

Residual connections



[Wu, et. al, Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation, 2016]()

Google Translate

How to train deep LSTM RNNs?

- The following relations govern the update between the i-th and i+1-th standard LSTM layers:

$$
\begin{aligned}
h_t^i, m_t^i &= LSTM_i(h_{t-1}^i, m_{t-1}^i, x_t^{i-1}, \mathbf{W}^i) \\
x_t^i &= h_t^i + x_t^{i-1} \quad \text{< residual connection} \\
h_t^{i+1}, m_t^{i+1} &= LSTM_{i+1}(h_{t-1}^{i+1}, m_{t-1}^{i+1}, x_t^i, \mathbf{W}^{i+1})
\end{aligned}
$$