Machine Learning

Lecture 08: Recurrent Neural Networks

Nevin L. Zhang
lzhang@cse.ust.hk

Department of Computer Science and Engineering
The Hong Kong University of Science and Technology

This set of notes is based on internet resources and references listed at the end.

Outline

Introduction

- So far, we have been talking about neural network models for labelled data:

$$\{\mathbf{x}_i, y_i\}_{i=1}^N \rightarrow P(y|\mathbf{x}),$$

  where each training example consists of **one input** $\mathbf{x}_i$ and **one output** y_i.

- Next, we will talk about neural network models for sequential data:

$$\{(\mathbf{x}_i^{(1)}, \ldots, \mathbf{x}_i^{(\tau_i)}), (\mathbf{y}_i^{(1)}, \ldots, \mathbf{y}_i^{(\tau_i)})\}_{i=1}^N \rightarrow P(\mathbf{y}^{(t)}|\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(t)}),$$

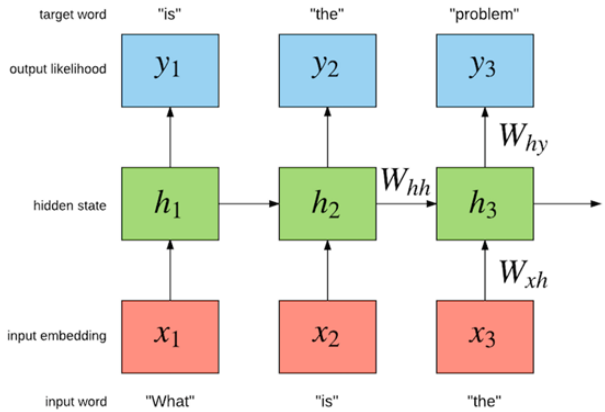
  where each training example consists of **a sequence of inputs** $(\mathbf{x}_i^{(1)}, \ldots, \mathbf{x}_i^{(\tau_i)})$ and **a sequence of outputs** $(\mathbf{y}_i^{(1)}, \ldots, \mathbf{y}_i^{(\tau_i)})$, and the current output $\mathbf{y}^{(t)}$ depends not only on the current input, but also all previous inputs.

Introduction: Language Modeling

- Data: A collection of sentences.
- For each sentence, create an output sequence by shifting it:

$$(\text{"}what\text{"}, \text{"}is\text{"}, \text{"}the\text{"}, \text{"}problem\text{"}), (\text{"}is\text{"}, \text{"}the\text{"}, \text{"}problem\text{"}, -)$$

- From the training pairs, we can learn a **neural language model**:



- It is used to predict the next word: $P(w_k|w_1, \ldots, w_{k-1})$.
- It also defines a probability distribution over sentences:

$$P(w_1, w_2, \ldots, w_n) = P(w_1)P(w_2|w_1)P(w_3|w_2, w_1)P(w_4|w_3, w_2, w_1)\ldots$$

Introduction: Dialogue and Machine Translation

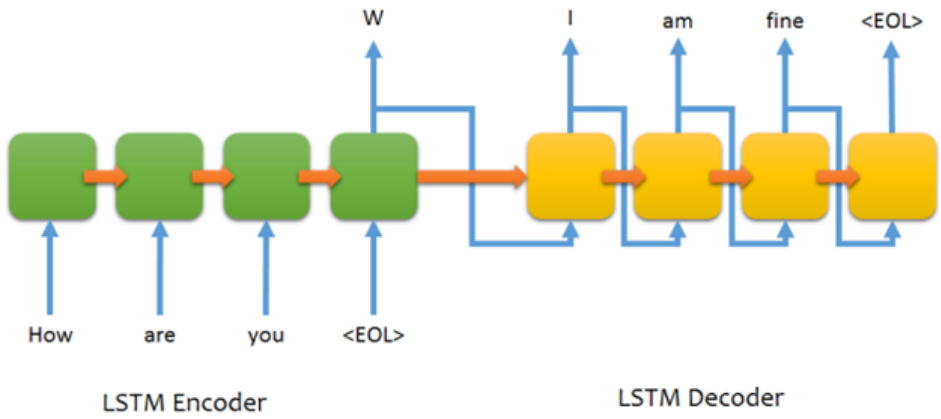
- Data: A collection of matched pairs.

$$\text{"How are you?"} \quad ; \text{"I am fine."}$$

- We can still thinking of having an input and an output at each time point, except some inputs and outputs are dummies.

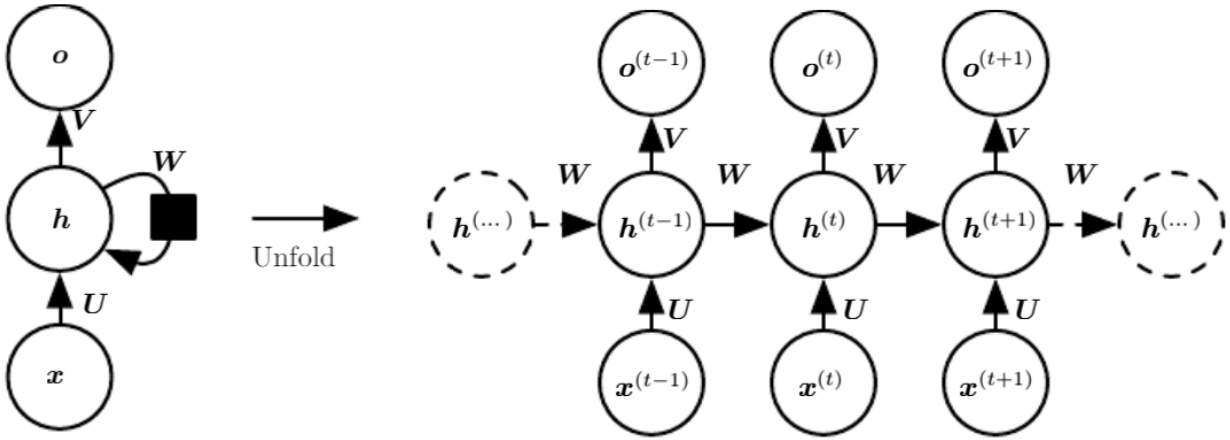
$$(\text{"}How\text{"}, \text{"}are\text{"}, \text{"}you\text{"}, -, -, -), (-, -, -, \text{"}I\text{"}, \text{"}am\text{"}, \text{"}fine\text{"}).$$

- From the training pairs, we can learn neural model for dialogue or machine translation.



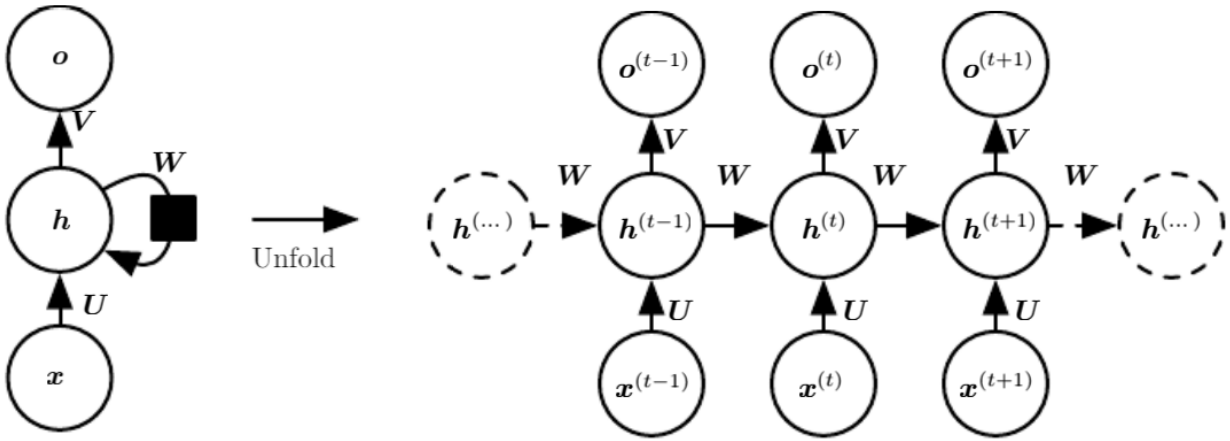
Outline

Recurrent Neural Networks



- A **circuit diagram**, aka **recurrent graph**, (left), where a black square indicates time-delayed dependence, or
- A **unfolded computational graph**, aka **unrolled graph**, (right). The length of the unrolled graph is determined by the length the input. In other words, the unrolled graphs for different sequences can be of different lengths.

Recurrent Neural Networks

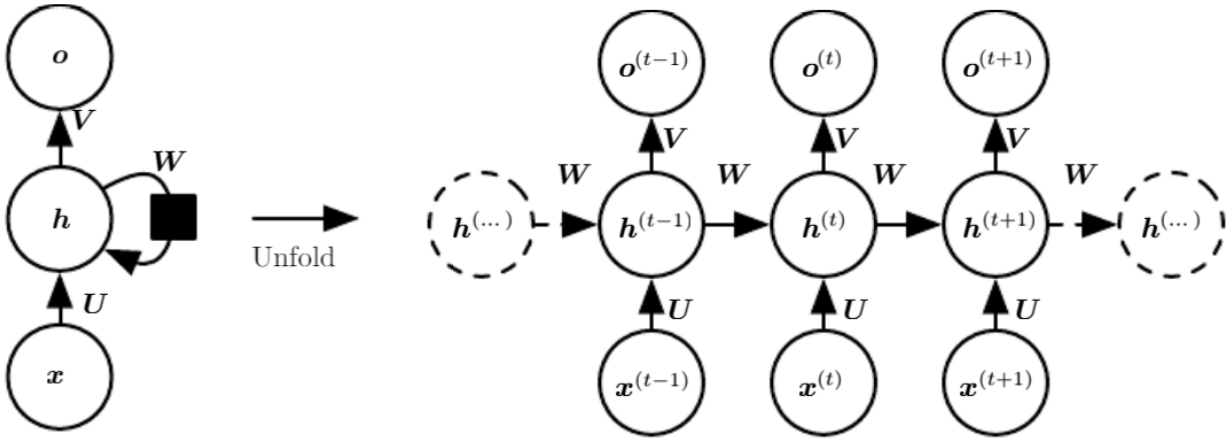


- The input tokens $\mathbf{x}^{(t)}$ are represented as **embedding vectors**, which are determined together with other model parameters during learning.
- The **hidden states** $\mathbf{h}^{(t)}$ are also vectors.
- The current state $\mathbf{h}^{(t)}$ depends on the current input $\mathbf{x}^{(t)}$ and the previous state $\mathbf{h}^{(t-1)}$ as follows:

$$\begin{aligned} \mathbf{a}^{(t)} &= \mathbf{b} + \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)} \\ \mathbf{h}^{(t)} &= \tanh(\mathbf{a}^{(t)}) \end{aligned}$$

where $\mathbf{b}$, $\mathbf{W}$ and $\mathbf{U}$ are model parameters. They are independent of time t.

Recurrent Neural Networks

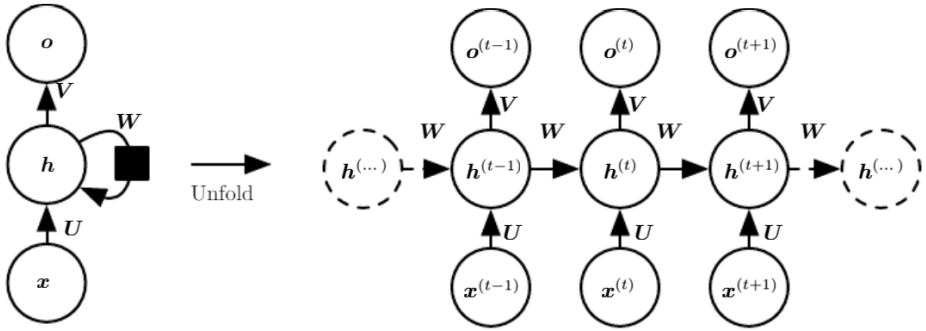


- The output sequence is produced as follows:

$$\begin{aligned} \mathbf{o}^{(t)} &= \mathbf{c} + \mathbf{V}\mathbf{h}^{(t)} \\ \hat{\mathbf{y}}^{(t)} &= \text{softmax}(\mathbf{o}^{(t)}) \end{aligned}$$

where $\mathbf{c}$ and $\mathbf{V}$ are model parameters. They are independent of time t.

Recurrent Neural Networks



- This is the **loss for one training pair**:

$$L(\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(\tau)}\}, \{\mathbf{y}^{(1)}, \dots, \mathbf{y}^{(\tau)}\}) = -\sum_{t=1}^{\tau} \log P_{\text{model}}(\mathbf{y}^{(t)} | \mathbf{x}^{(1)}, \dots, \mathbf{x}^{(t)}),$$

  where $\log P_{\text{model}}(\mathbf{y}^{(t)} | \mathbf{x}^{(1)}, \dots, \mathbf{x}^{(t)})$ is obtained by reading the entry for $\mathbf{y}^{(t)}$ from the models output vector $\hat{\mathbf{y}}^{(t)}$.

- When there are multiple input-target sequence pairs, the losses are added up.
- Training objective: Minimize the **total loss of all training pairs** w.r.t the model parameters and embedding vectors:

$$\mathbf{W}, \mathbf{U}, \mathbf{V}, \mathbf{b}, \mathbf{c}, \theta_{em}$$

  where θ_{em} are the embedding vectors.

Training RNNs

- RNNs are trained using stochastic gradient descent.
- We need gradients:

$$\nabla_{\mathbf{W}} L, \nabla_{\mathbf{U}} L, \nabla_{\mathbf{V}} L, \nabla_{\mathbf{b}} L, \nabla_{\mathbf{c}} L, \nabla_{\theta_{em}} L$$

  They are computed using **Backpropagation Through Time (BPTT)**, which is an adaption of Backpropagation to the unrolled computational graph.
- BPTT is implemented in deep learning packages such as Tensorflow.

RNN and Self-Supervised Learning

- **Self-supervised learning** is a learning technique where the training data is automatically labelled.
- It is still supervised learning, but the **datasets do not need to be manually labelled by a human**, but they can e.g. be labelled by finding and exploiting the relations between different input signals.
- RNN training is self-supervised learning.

Outline

Basic Idea

- **Long Short-Term Memory (LSTM)** unit is a widely used technique to address long-term dependencies.
- The key idea is to use **memory cells** and **gates**:

$$c_{(t)} = f_t c_{(t-1)} + i_t a^{(t)}$$

  - $c_{(t)}$: memory state at t; $a^{(t)}$: new input at t.
  - If the **forget gate** f_t is open (i.e, 1) and the **input gate** i_t is closed (i.e., 0), the current memory is kept.
  - If the **forget gate** f_t is closed (i.e, 0) and the **input gate** i_t is open (i.e., 1), the current memory is erased and replaced by new input.
  - If we can learn f_t and i_t from data, then we can automatically determine how much history to remember/forget.

Basic Idea

- In the case of vectors, $\mathbf{c}_{(t)} = \mathbf{f}_t \otimes \mathbf{c}_{(t-1)} + \mathbf{i}_t \otimes \mathbf{a}^{(t)}$, where $\otimes$ means pointwise product.
- $\mathbf{f}_t$ is called the **forget gate** vector because it determines which components of the previous state and how much of them to remember/forget.
- $\mathbf{i}_t$ is called the **input gate** vector because it determines which components of the input from $\mathbf{h}^{(t-1)}$ and $\mathbf{x}^{(t)}$ and how much of them should go into the current state.
- If we can learn $\mathbf{f}_t$ and $\mathbf{i}_t$ from data, then we can automatically determine which component to remember/forget and how much of them to remember/forget.

LSTM Cell

- In standard RNN, $\mathbf{a}^{(t)} = \mathbf{b} + \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)}, \mathbf{h}^{(t)} = \tanh(\mathbf{a}^{(t)})$
- In LSTM, we introduce a **cell state** vector $\mathbf{c}_t$, and set

$$\begin{aligned} \mathbf{c}_t &= \mathbf{f}_t \otimes \mathbf{c}_{t-1} + \mathbf{i}_t \otimes \tanh(\mathbf{a}^{(t)}) \\ \mathbf{h}^{(t)} &= \tanh(\mathbf{c}_t) \end{aligned}$$

  where $\mathbf{f}_t$ and $\mathbf{i}_t$ are vectors.

LSTM Cell: Learning the Gates

- $\mathbf{f}_t$ is determined based on current input $\mathbf{x}^{(t)}$ and previous hidden unit $\mathbf{h}^{(t-1)}$:

$$\mathbf{f}_t = \sigma(\mathbf{W}_f \mathbf{h}^{(t-1)} + \mathbf{U}_f \mathbf{x}^{(t)} + \mathbf{b}_f),$$

  where $\mathbf{W}_f$, $\mathbf{U}_f$, $\mathbf{b}_f$ are parameters to be learned from data.

- $\mathbf{i}_t$ is also determined based on current input $\mathbf{x}^{(t)}$ and previous hidden unit $\mathbf{h}^{(t-1)}$:

$$\mathbf{i}_t = \sigma(\mathbf{W}_i \mathbf{h}^{(t-1)} + \mathbf{U}_i \mathbf{x}^{(t)} + \mathbf{b}_i)$$

  where $\mathbf{W}_i$, $\mathbf{U}_i$, $\mathbf{b}_i$ are parameters to be learned from data.

- Note the sigmoid activation function is used for the gates so that their values are often close to 0 or 1. In contrast, tanh is used for the output $\mathbf{h}^{(t)}$ so as to have strong gradient signal during backprop.

LSTM Cell: Output Gate

- We can also have a **output gate** to control which components of the state vector $\mathbf{c}_t$ and how much of them should be outputted:

$$\mathbf{o}_t = \sigma(\mathbf{W}_o \mathbf{h}^{(t-1)} + \mathbf{U}_o \mathbf{x}^{(t)} + \mathbf{b}_o)$$

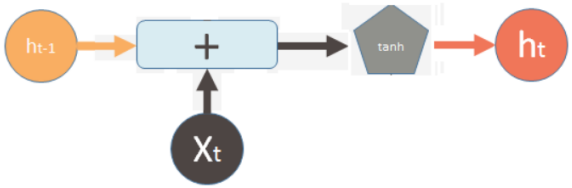
  where $\mathbf{W}_q$, $\mathbf{U}_q$, $\mathbf{b}_q$ are the learnable parameters, and set

$$\mathbf{h}^{(t)} = \mathbf{o}_t \otimes \tanh(\mathbf{c}_t)$$

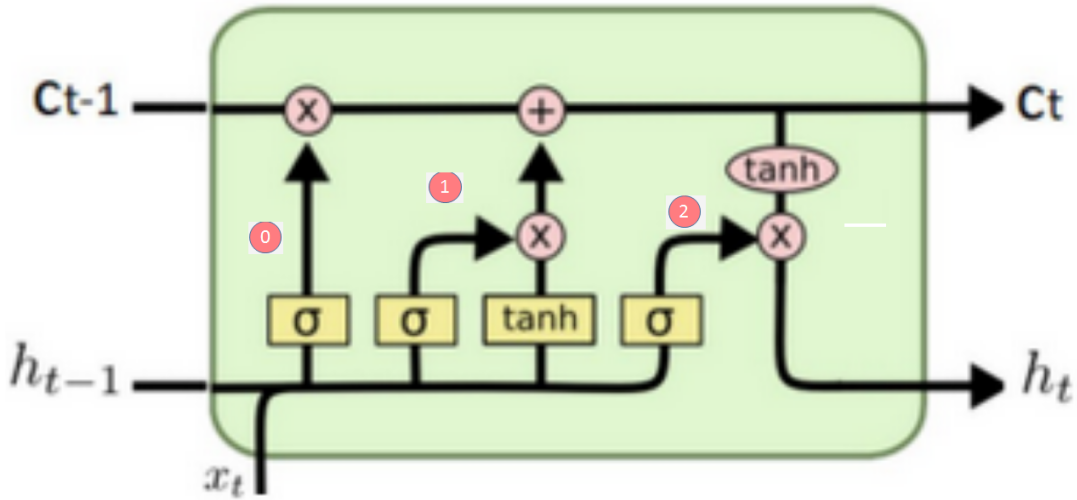
LSTM Cell: Summary

- A **standard RNN cell**: $\mathbf{a}^{(t)} = \mathbf{b} + \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{U}\mathbf{x}^{(t)}, \mathbf{h}^{(t)} = \tanh(\mathbf{a}^{(t)})$
- An **LSTM Cell**:

$$
\begin{aligned}
\mathbf{f}_t &= \sigma(\mathbf{W}_f\mathbf{h}^{(t-1)} + \mathbf{U}_f\mathbf{x}^{(t)} + \mathbf{b}_f) && \text{(forget gate, ⓪ in figure)} \\
\mathbf{i}_t &= \sigma(\mathbf{W}_i\mathbf{h}^{(t-1)} + \mathbf{U}_i\mathbf{x}^{(t)} + \mathbf{b}_i) && \text{(input gate, ① in figure)} \\
\mathbf{o}_t &= \sigma(\mathbf{W}_o\mathbf{h}^{(t-1)} + \mathbf{U}_o\mathbf{x}^{(t)} + \mathbf{b}_o) && \text{(output gate, ③ in figure)} \\
\mathbf{c}_t &= \mathbf{f}_t \otimes \mathbf{c}_{t-1} + \mathbf{i}_t \otimes \tanh(\mathbf{U}\mathbf{x}^{(t)} + \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{b}) && \text{(update memory)} \\
\mathbf{h}^{(t)} &= \mathbf{o}_t \otimes \tanh(\mathbf{c}_t) && \text{(next hidden unit)}
\end{aligned}
$$



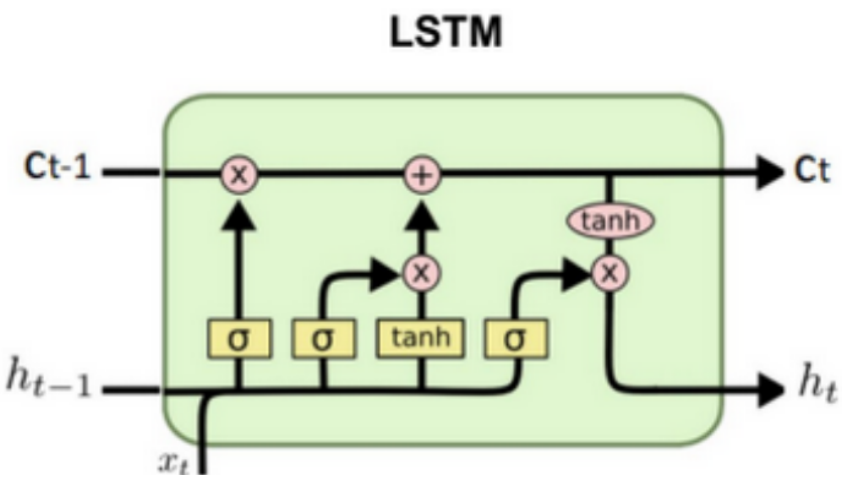
Standard RNN Cell

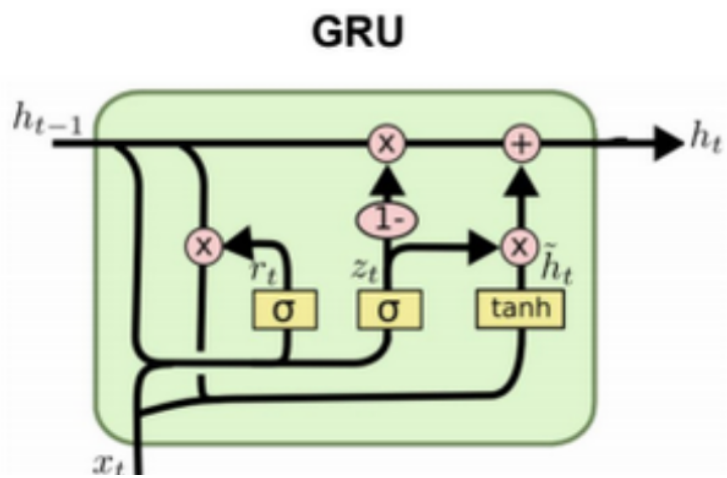


LSTM Cell

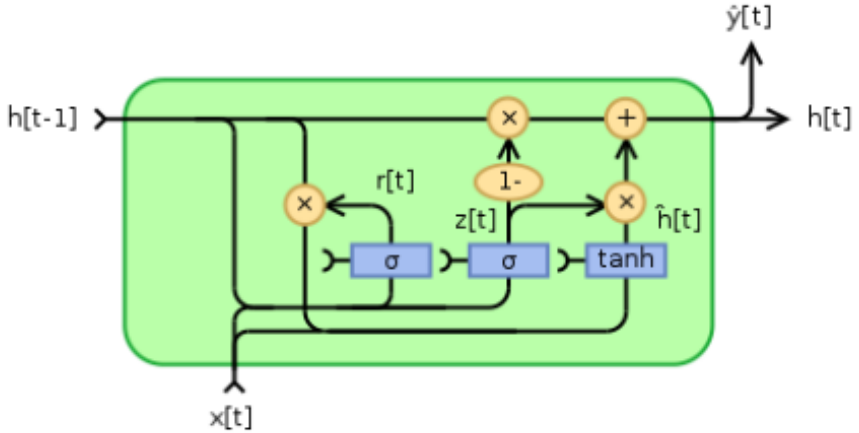
Gated Recurrent Unit

- The Gated Recurrent Unit (GRU) is another a gating mechanism to allow RNNs to efficiently learn long-range dependency.
- It is similar to LSTM (no memory and no output unit) and hence has fewer paramters. is a simplified version of an LSTM unit with fewer parameters.
- Performance also similar to LSTM, except better on small datasets.





Gated Recurrent Unit



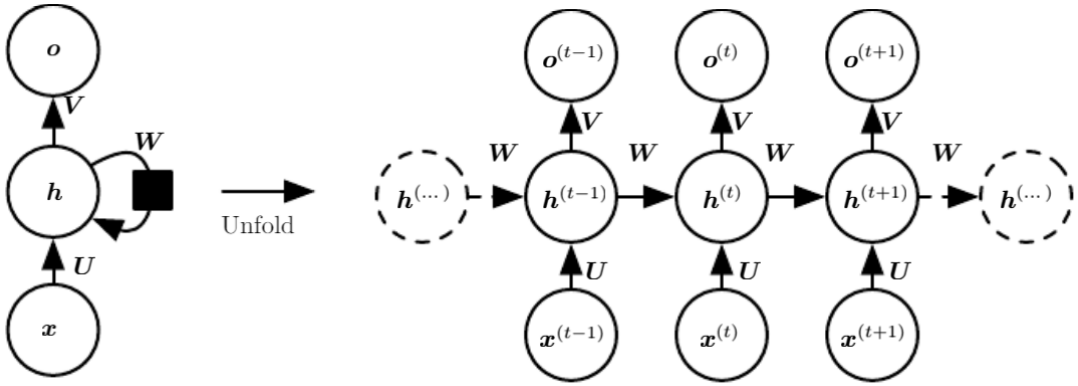
$$z_t = \sigma_g(W_z x_t + U_z h_{t-1} + b_z)$$
$$r_t = \sigma_g(W_r x_t + U_r h_{t-1} + b_r)$$
$$h_t = (1 - z_t) \circ h_{t-1} + z_t \circ \sigma_h(W_h x_t + U_h(r_t \circ h_{t-1}) + b_h)$$

- x_t: input vector
- z_t: update gate vector
- h_t: output vector
- r_t: reset gate vector
- W, U and b: parameter matrices and vector

Outline

So Far ....

- So far, we have concentrated on the following architecture that can be used to model the relationship pairs of sequences $(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(\tau)})$ and $(\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(\tau)})$ of the same length.
- It is used for language modeling.



Deep RNNs

- Sometimes, we might want to use multiple layers of hidden units. This leads to **deep recurrent neural network**.
- More layers usually implies better performance.

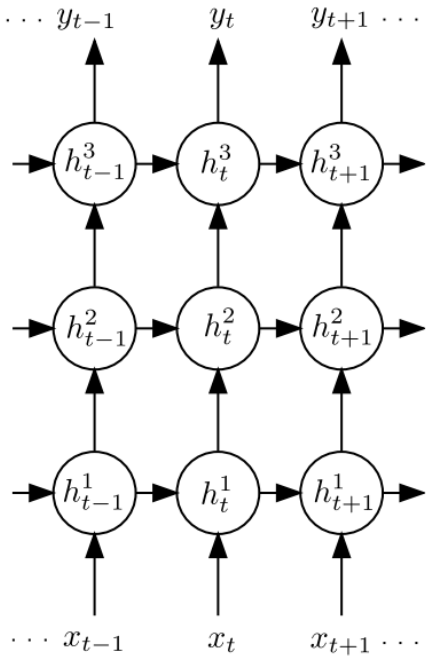
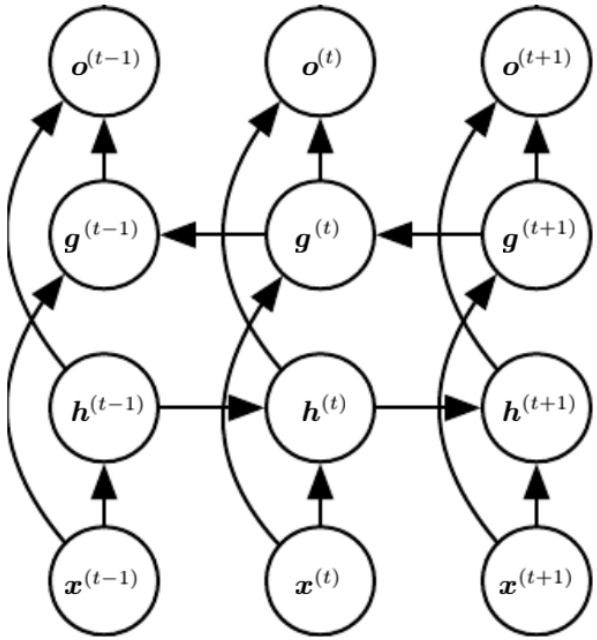


Fig. 3. Deep Recurrent Neural Network

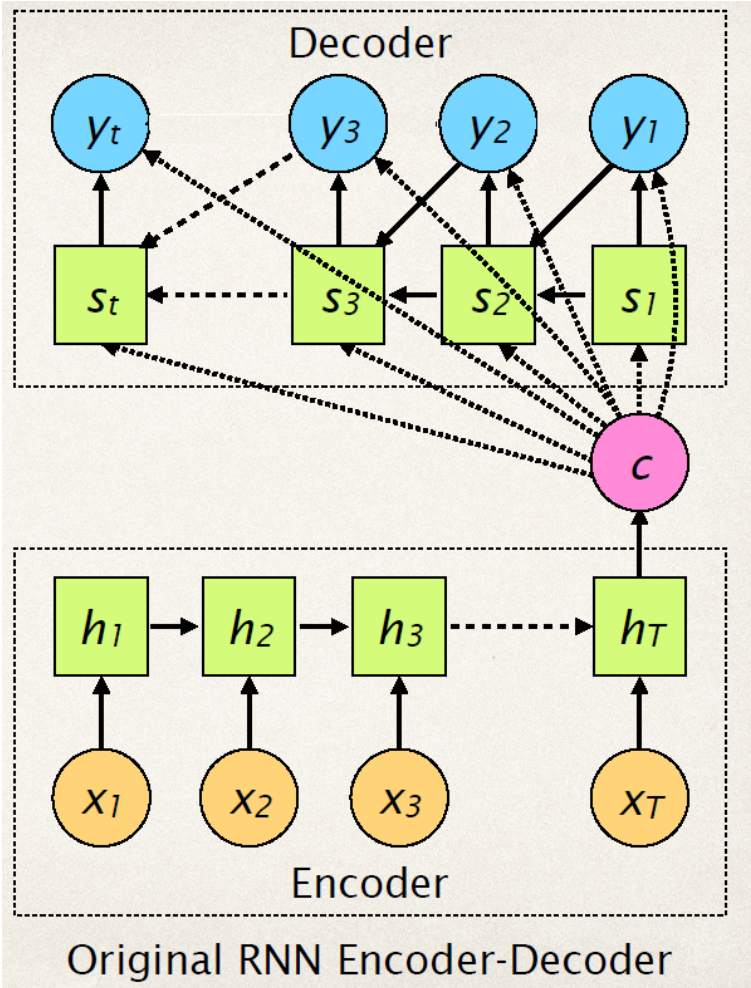
Bidirectional RNNs

- In one directional RNN, $\mathbf{h}^{(t)}$ only capture information from the past $(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(t)})$, and we use $\mathbf{h}^{(t)}$ to predict $\mathbf{y}^{(t)}$
- Sometime, we might need information from the future to predict $\mathbf{y}^{(t)}$.
- In such case, we can use bidirectional RNNs.
- Found useful in handwriting recognition, speech recognition, and bioinformatics.



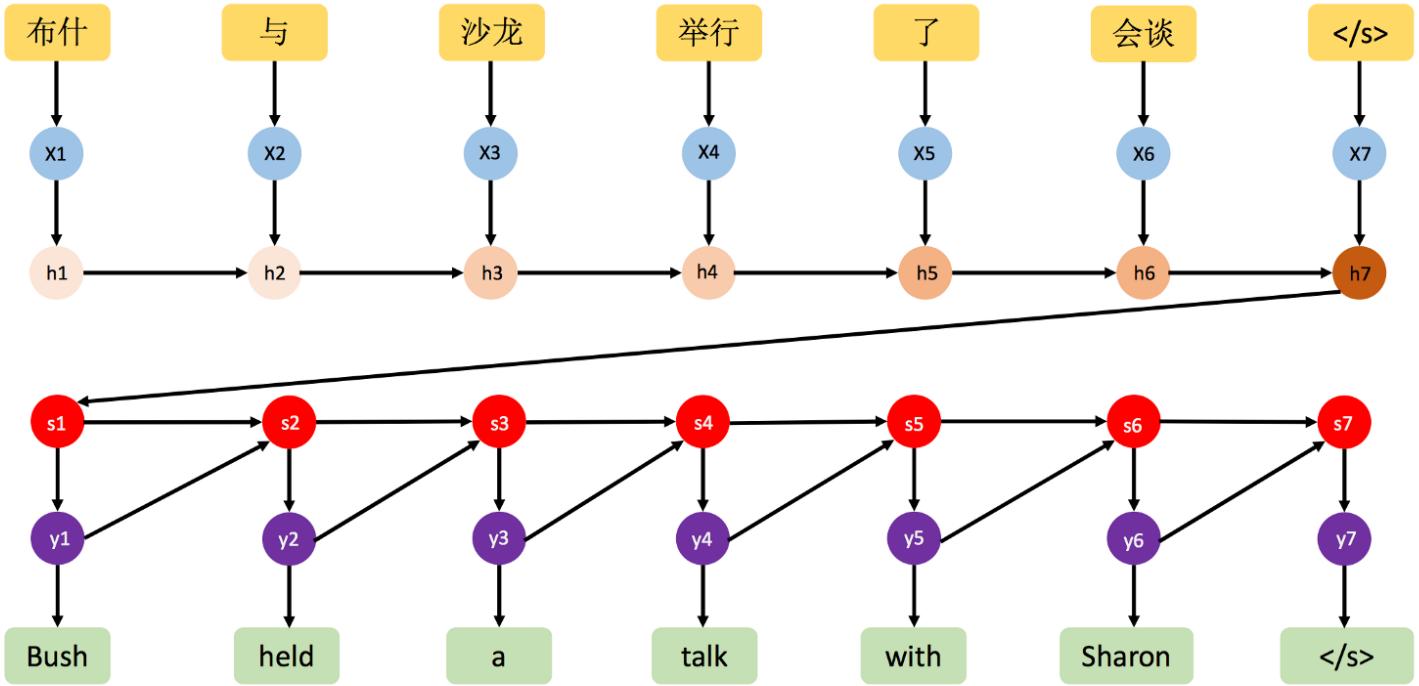
The Encoder-Decoder Architecture

- The **encoder-decoder** or **sequence to sequence** architecture is used for learning to generate a output sequence $(\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(n_y)})$ in response to an input sequence $(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n_x)})$.
- The **context variable** C represents a semantic summary of the input the sequence.
- The decoder defines a conditional distribution $P(\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(n_y)}|C)$ over sequences, from which output sequences are generated.
- The architecture is used in dialogue systems and machine translation.



Seq2seq for machine translation

- The generated sequence can be a translation of the input sequence in machine translation,

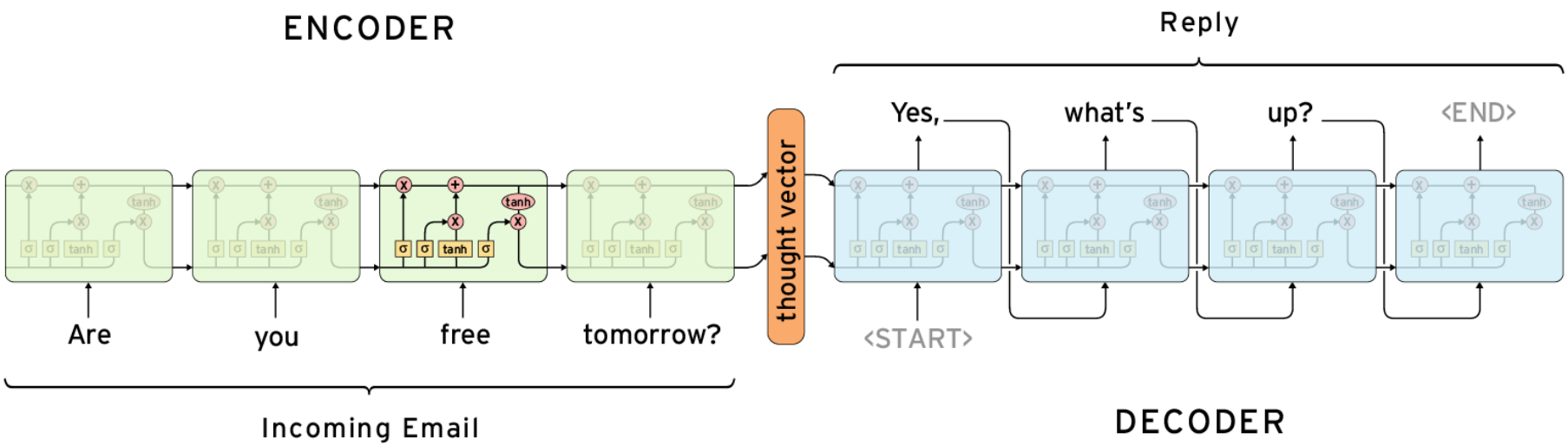


(Sutskever et al., 2014)

Note that this model assumes: $p(\mathbf{s}_2|\mathbf{C}, \mathbf{s}_1, \mathbf{y}^{(1)}) = p(\mathbf{s}_2|\mathbf{s}_1, \mathbf{y}^{(1)}), \ldots$

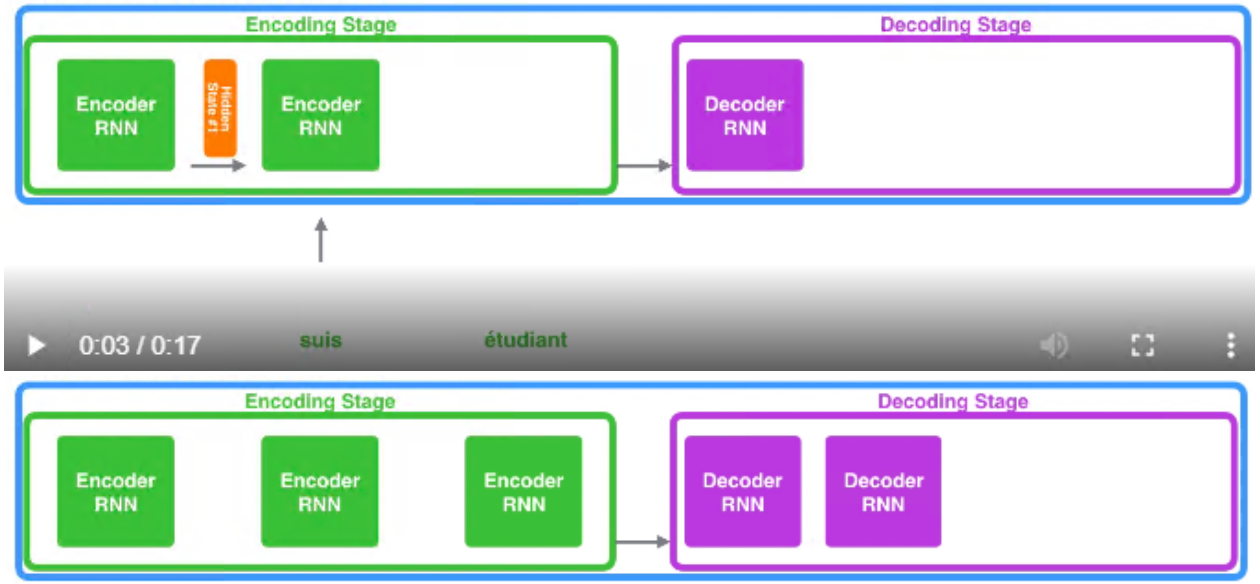
Seq2seq for ChatBot

- The generated sequence can also be a reply to the input sequence in dialogue systems



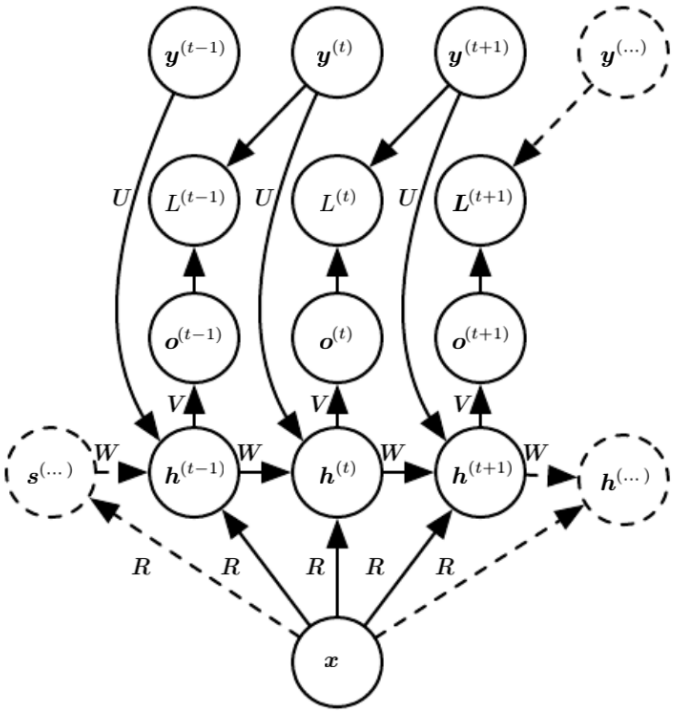
Seq2seq in Action

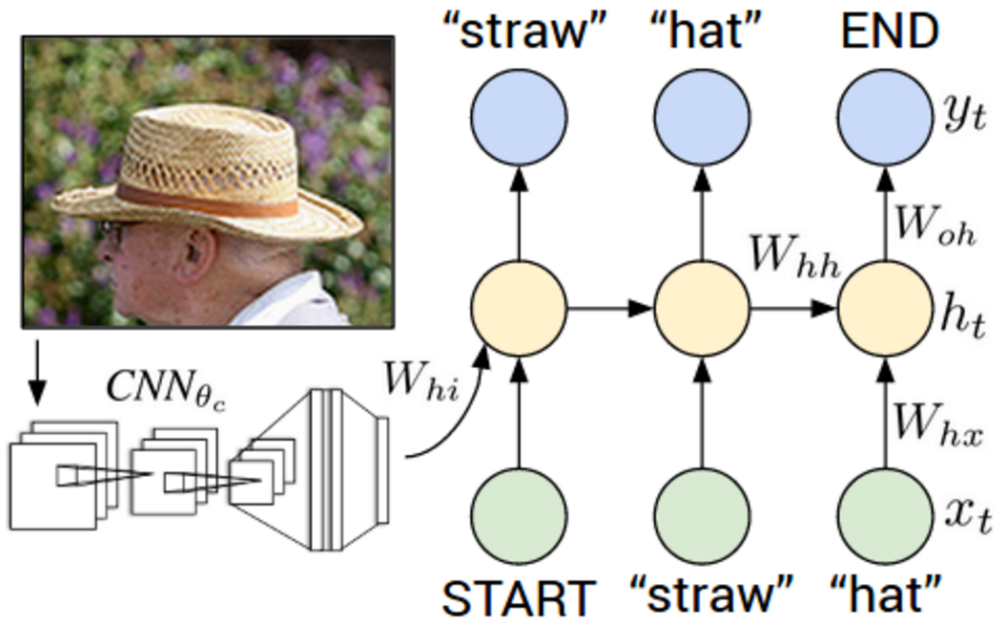
http://jalammar.github.io/images/seq2seq_6.mp4



Map a Vector to A sequence

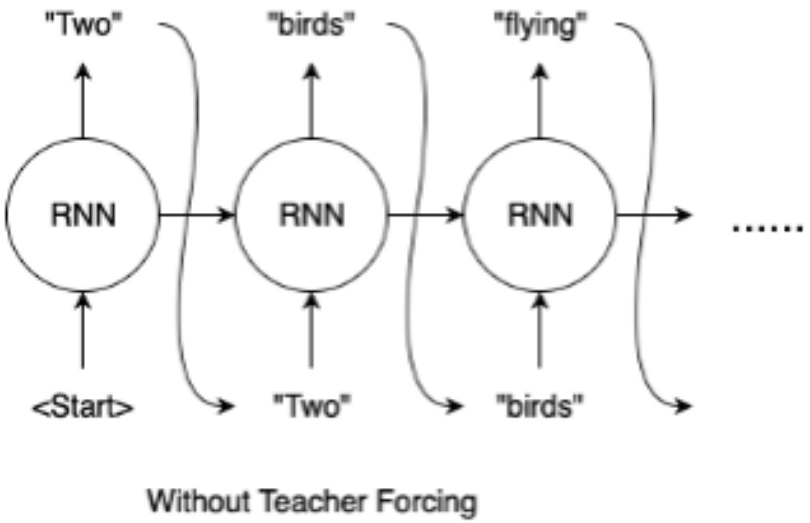
- Maps a vector $\mathbf{x}$ to a sequence $(\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(\tau)})$
- Found useful in image captioning



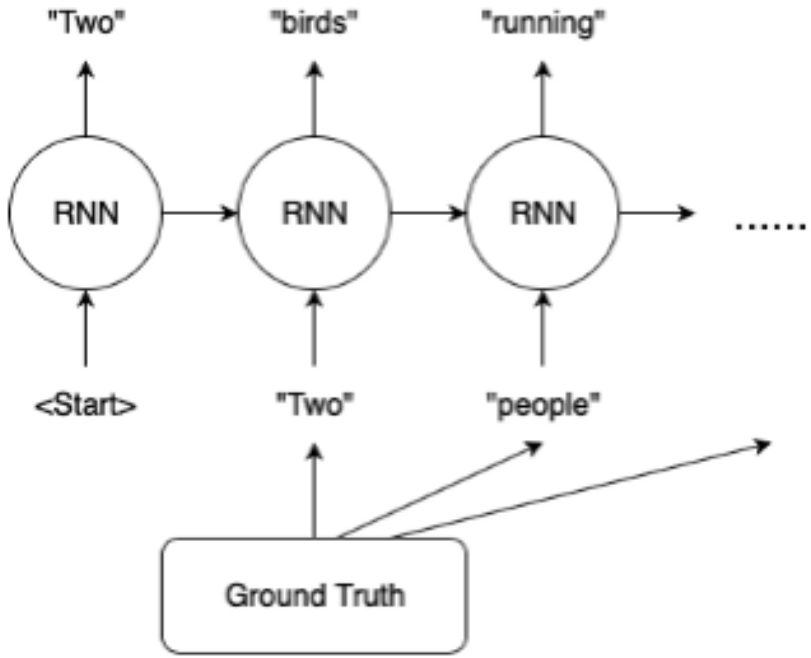


Teacher Forcing in Decoder

- Suppose ground-truth output is: "Two people reading a book".
- But model makes mistake at the second position.



Without Teacher Forcing



With Teacher Forcing

- **Teacher Forcing**, feed "people" to our RNN for the 3rd prediction, after computing and recording the loss for the 2nd prediction.

Teacher Forcing in Decoder

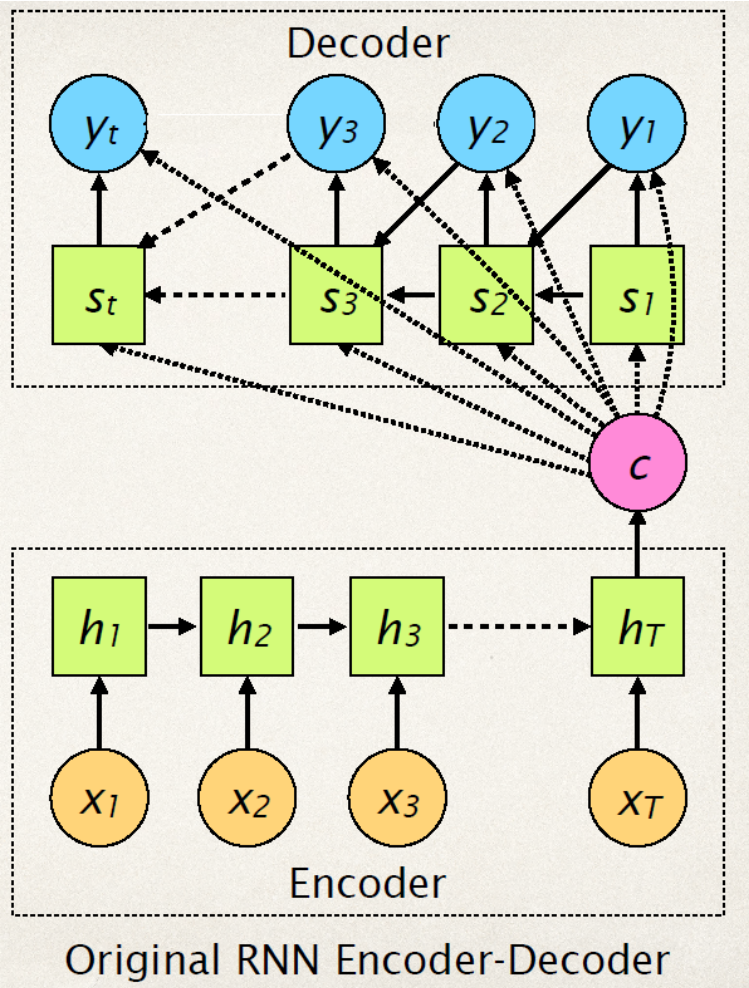
- **Pros:** Training with Teacher Forcing converges faster. At the early stages of training, the predictions of the model are very bad. If we do not use Teacher Forcing, the hidden states of the model will be updated by a sequence of wrong predictions, errors will accumulate, and it is difficult for the model to learn from that.
- **Cons:** During inference, since there is usually no ground truth available, the RNN model will need to feed its own previous prediction back to itself for the next prediction. Therefore there is a discrepancy between training and inference, and this might lead to poor model performance and instability. This is known as Exposure Bias in literature.

- Recent work has shown that exposure bias is not a serious issue.

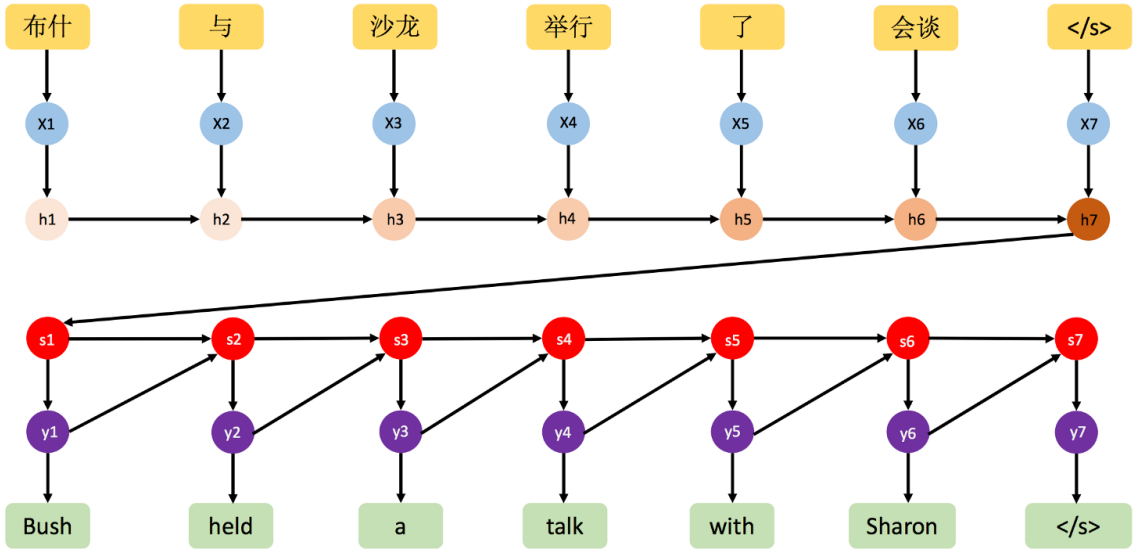
Outline

Motivation

- In the seq2seq model, the context variable **C** is shared at all time steps of the decoder.
- So, we are look at the **same summary** of the input sequence when generating the output **at all time steps**.
- This is not desirable.



Motivation



(Sutskever et al., 2014)

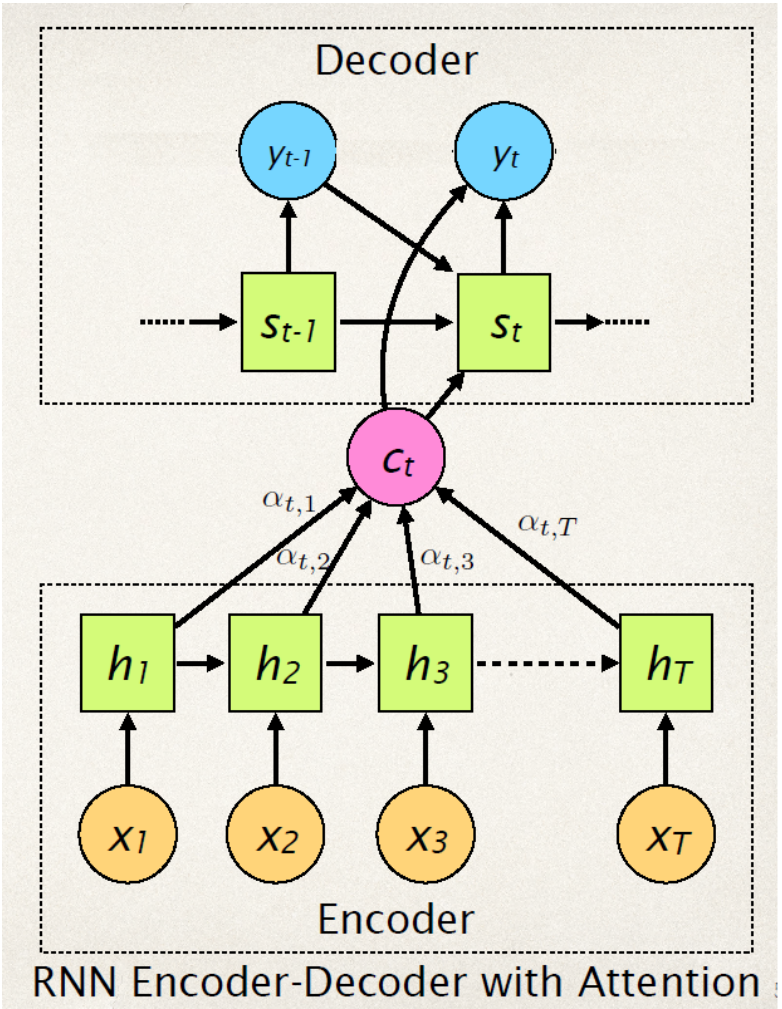
- In fact, we might want to focus on **different parts** of the input **at different times**:
  - Focus at $\mathbf{h}_1$ at time step 1,
  - Focus at $\mathbf{h}_4$ at time step 2,
  - Focus at $\mathbf{h}_6$ at time step 4,
  - ...

Attention

- So, we introduce a context variable $\mathbf{C}_t$ for each time step of the decoder:

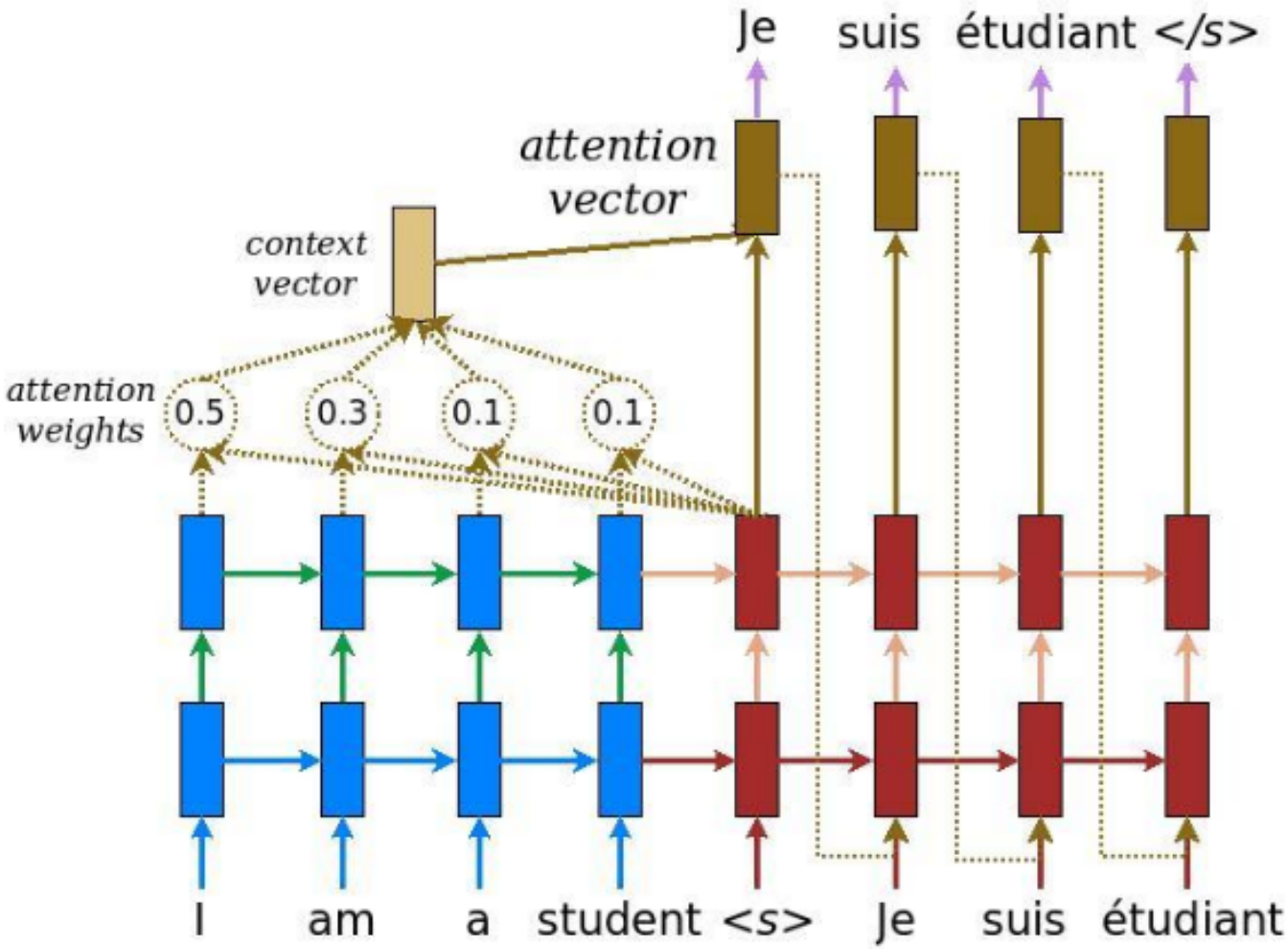
$$\mathbf{C}_t = \sum_j \alpha_{t,j} \mathbf{h}_j$$

- It is a linear combinations of the hidden states from the encoder.
- At different time steps t, we use different values for the weights $\alpha_{t,j}$, and consequently focus on different parts of the input.



RNN Encoder-Decoder with Attention

Attention Example



Attention

For a particular sequence, we have finished step $t-1$ of decoding. What information do we use to determine the attention weights $\alpha_{t,j}$ for step t?

- $\mathbf{c}_{t-1}$: Context vectors for step $t-1$. It tells us what to look for next.
  - If "held" attended to at step $t-1$, likely to have "talk", "meeting", etc at step t.
  - If "drink" attended to at step $t-1$, likely to have "water", "juice", etc at step t.

  In general, $\mathbf{c}_{t-1}$ is known as the **query**.
- $\mathbf{h}_j$: Latent state of step j of encoding. It tells us whether $\mathbf{h}_j$ is what we need next. In general, it is known as the **key**.

The attention weight is to be determined by assessing how compatible the query and the key are.

Compatibility Function

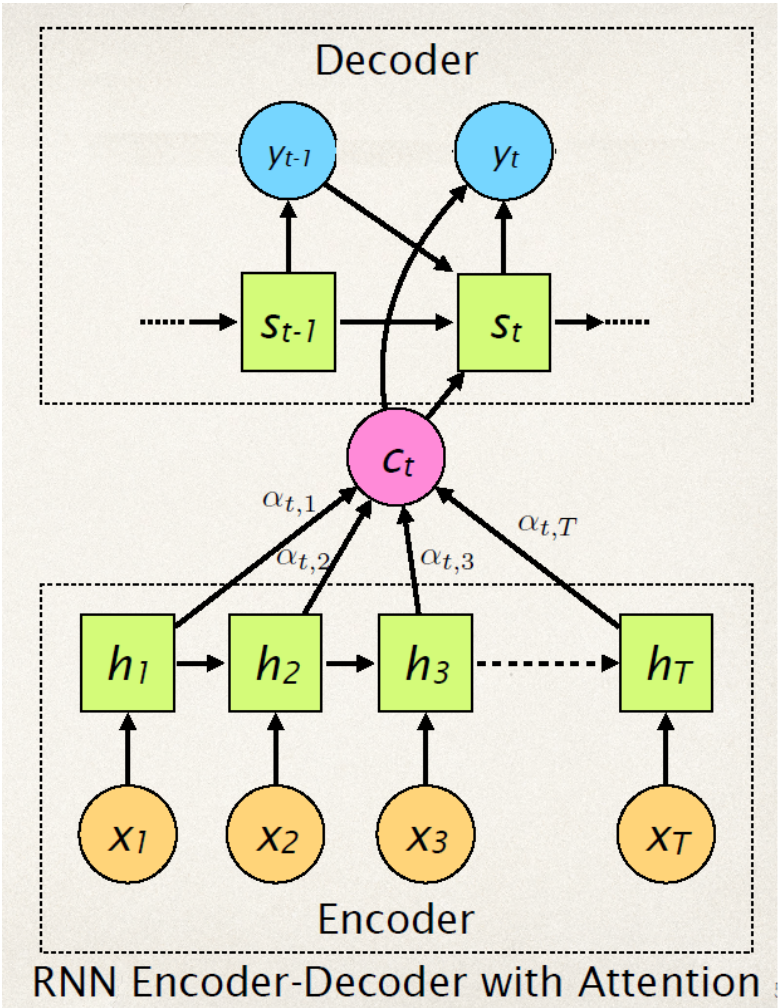
- $\alpha_{t,j} = \mathbf{h}_j^\top \mathbf{c}_{t-1}$, assuming $\mathbf{h}_j$ and $\mathbf{c}_{t-1}$ have equal dimensionality. This is called **dot-product attention)**.
- $\alpha_{t,j} \leftarrow \frac{exp(\alpha_{t,j})}{\sum_j exp(\alpha_{t,j})}$

Attention

- The weight $\alpha_{t,j}$ is obtained using the query $\mathbf{c}_{t-1}$ and the key $\mathbf{h}_j$?
  - It tells us how relevant the information at step j.
- Then we retrieve the **value** $\mathbf{h}_j$ at step j and computed their weighted sum to get:

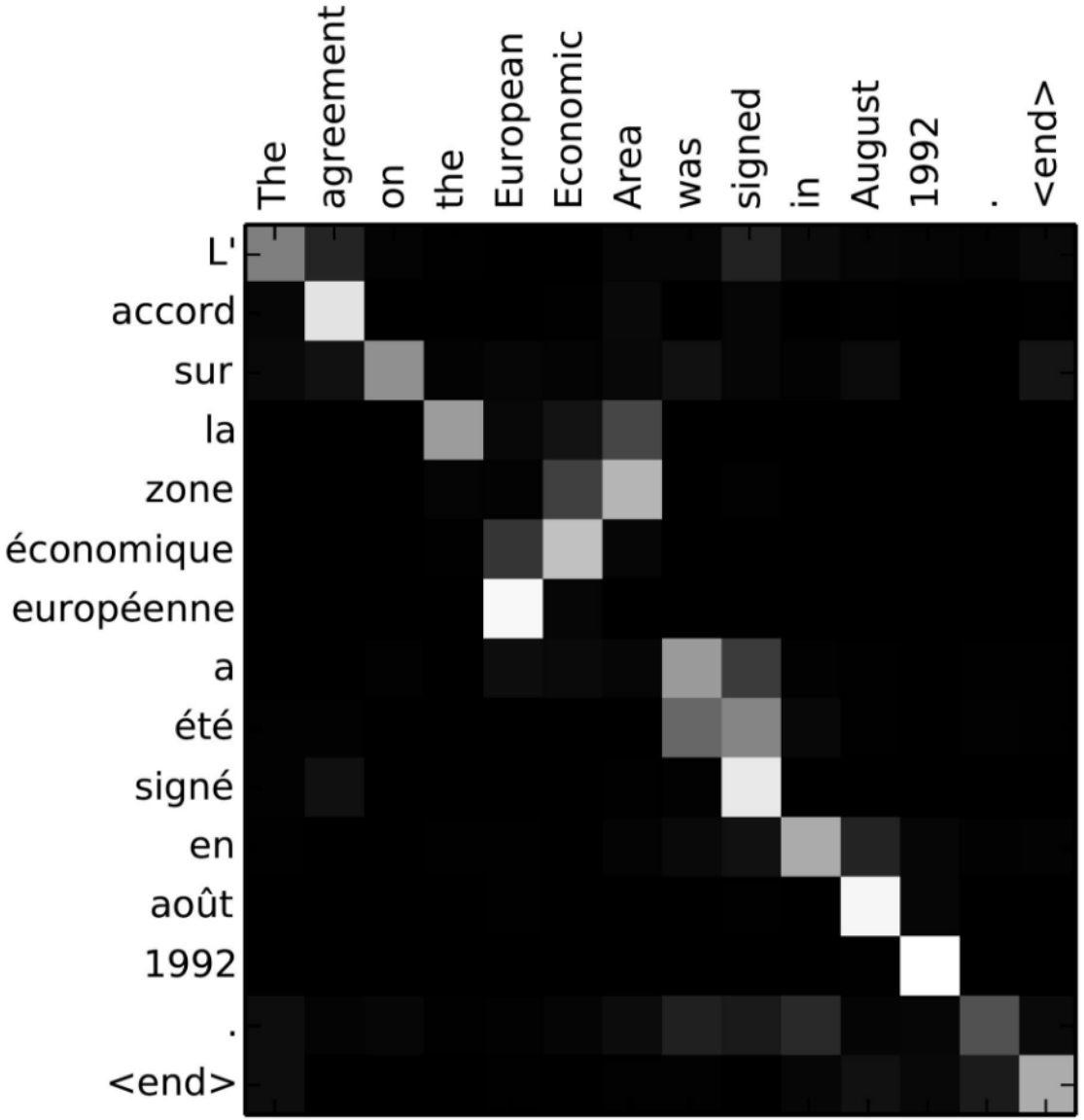
$$\mathbf{C}_t = \sum_j \alpha_{t,j} \mathbf{h}_j$$

- Note that, for different input sequences, the weights $\alpha_{t,j}$ are different. So, attention means **dynamic weights**. (In standard NN, weights are static and do not depend on input.)



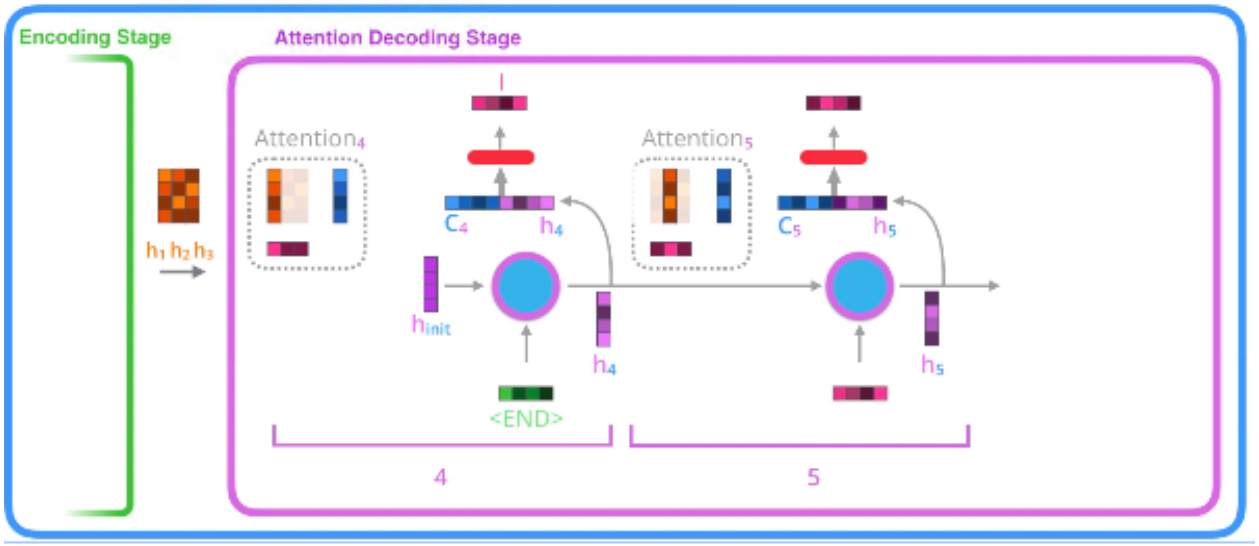
RNN Encoder-Decoder with Attention

Alignments between source and target sentences



Attention in Action

http://jalammar.github.io/images/attention_tensor_dance.mp4



References

- Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate." arXiv preprint arXiv:1409.0473 (2014).
- Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." arXiv preprint arXiv:1810.04805 (2018).
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT press.`www.deeplearningbook.org`
- Jay Alammar: `http://jalammar.github.io/`
- Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems. 2017.