Precept 7: seq2seq, attention, and transformers

Samyak Gupta

04/7/2023

PSA

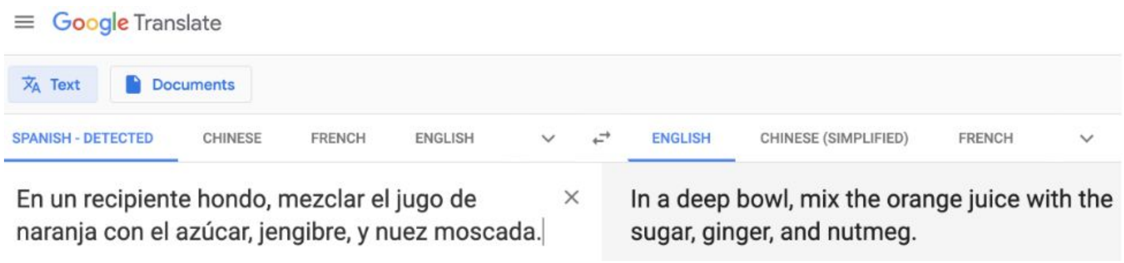
Start assignment 4 early

It’s more involved than the previous ones

Agenda

- seq2seq
- Attention
- Transformers

Machine Translation

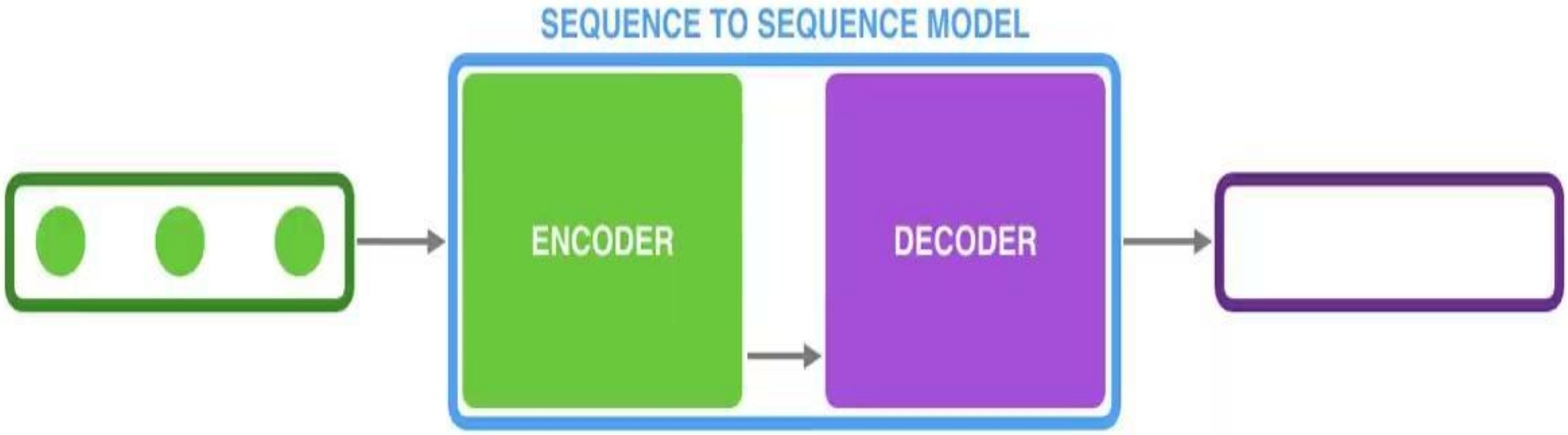


“Source” language → “Target” language

Difficult due to nuances of language

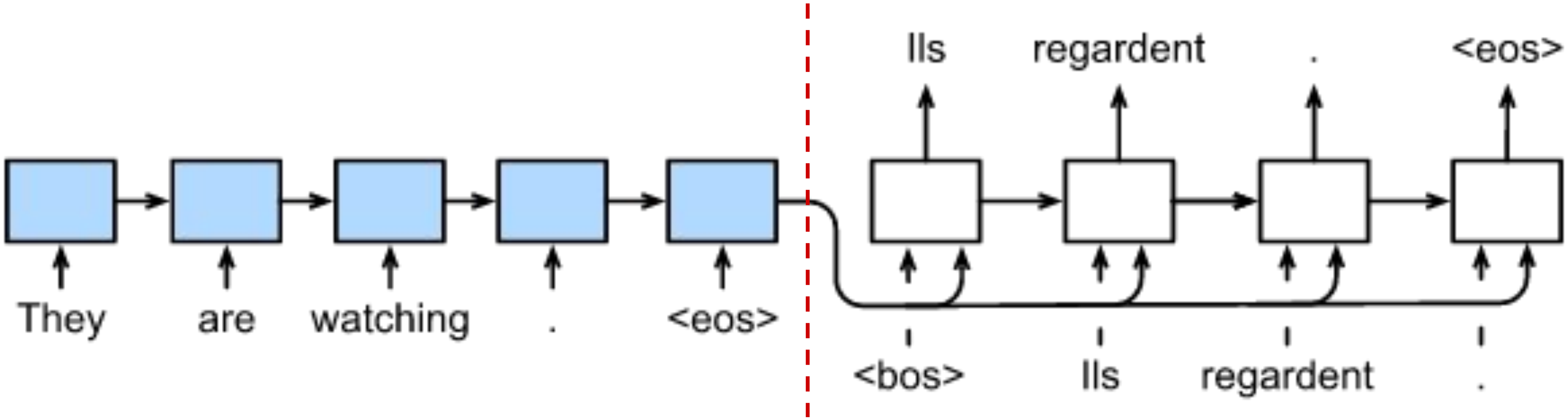
seq2seq models

Goal: Transform from a source sequence to a target sequence



seq2seq (with RNNs) for machine translation

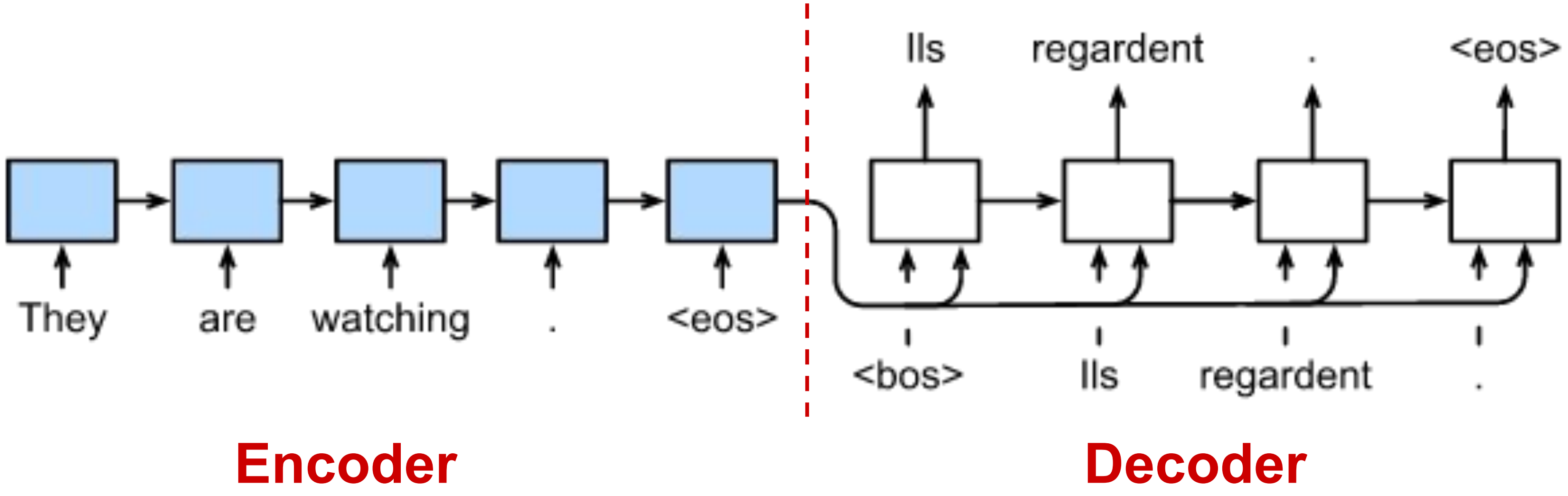
Key idea: use two RNNs



Encoder **Decoder**

seq2seq (with RNNs) for machine translation

Key idea: use two RNNs



(In assignment 4, your encoder and decoder will be based on transformers instead of RNNs)

seq2seq encoder

Encoder: Transform some source sequence into a hidden representation

seq2seq encoder

Encoder: Transform some source sequence into a hidden representation

Sentence: hello world .

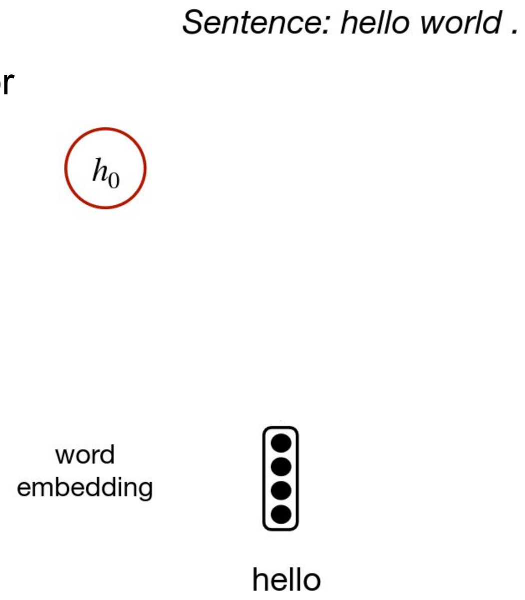
Step 1: Transform word to a vector (using embeddings matrix)

hello

seq2seq encoder

Encoder: Transform some source sequence into a hidden representation

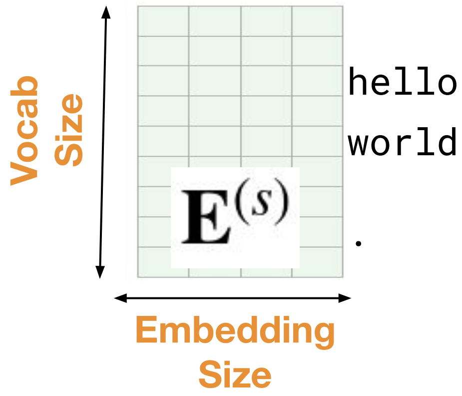
Step 1: Transform word to a vector (using embeddings matrix)



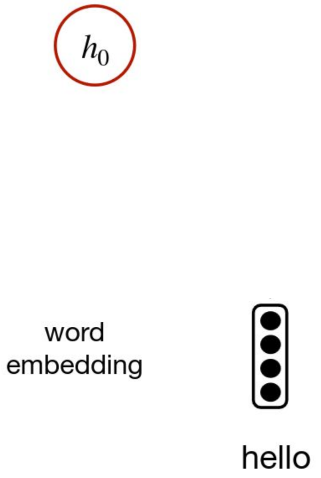
seq2seq encoder

Encoder: Transform some source sequence into a hidden representation

Step 1: Transform word to a vector (using embeddings matrix $\mathbf{E}^{(s)}$)



Sentence: hello world .

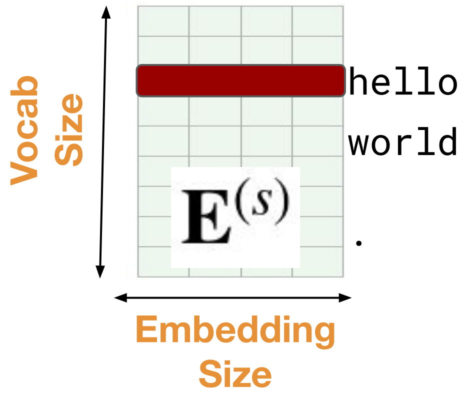


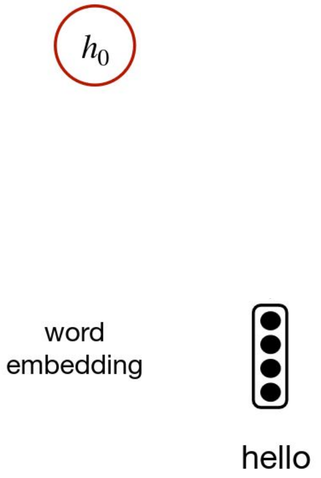
seq2seq encoder

Encoder: Transform some source sequence into a hidden representation

Sentence: hello world .

Step 1: Transform word to a vector (using embeddings matrix $\mathbf{E}^{(s)}$)



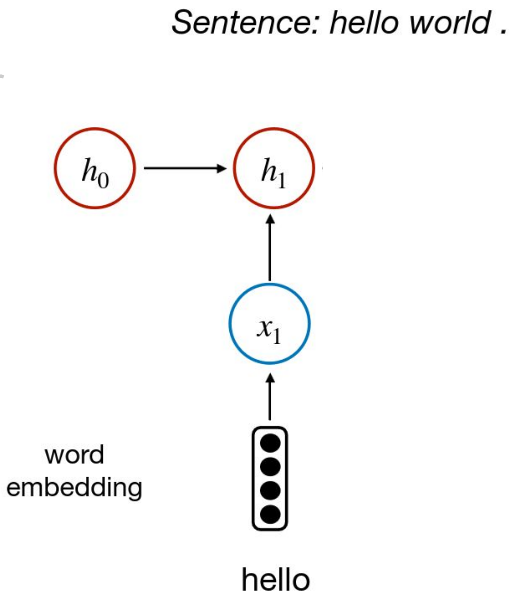


seq2seq encoder

Encoder: Transform some source sequence into a hidden representation

Step 1: Transform word to a vector (using embeddings matrix)

Step 2: Compute hidden state using word embedding and last hidden state

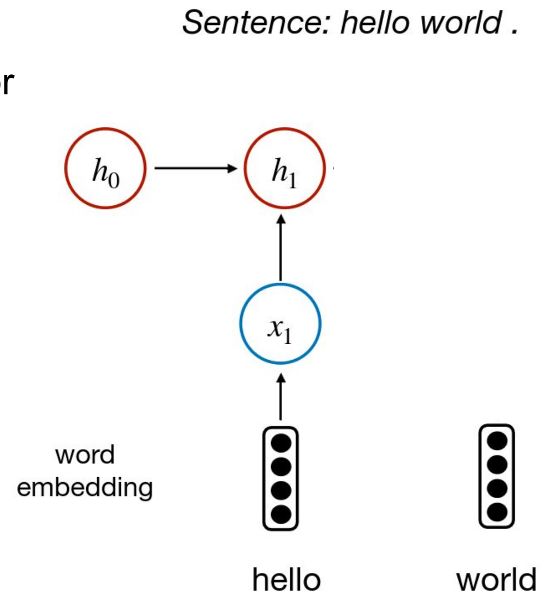


seq2seq encoder

Encoder: Transform some source sequence into a hidden representation

Step 1: Transform word to a vector (using embeddings matrix)
Step 2: Compute hidden state using word embedding and last hidden state

Repeat!

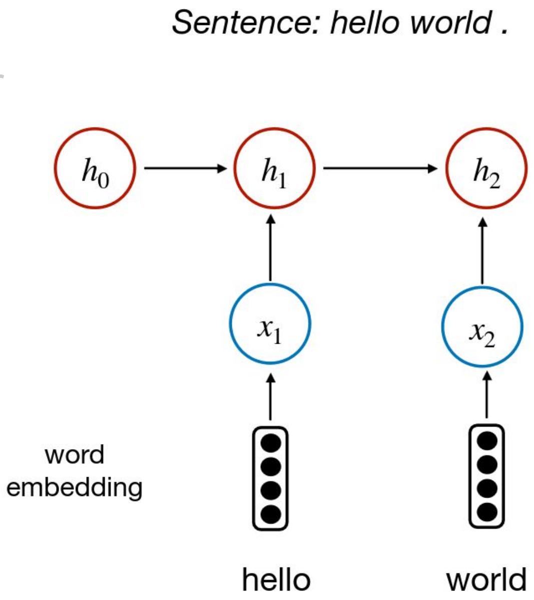


seq2seq encoder

Encoder: Transform some source sequence into a hidden representation

Step 1: Transform word to a vector (using embeddings matrix)

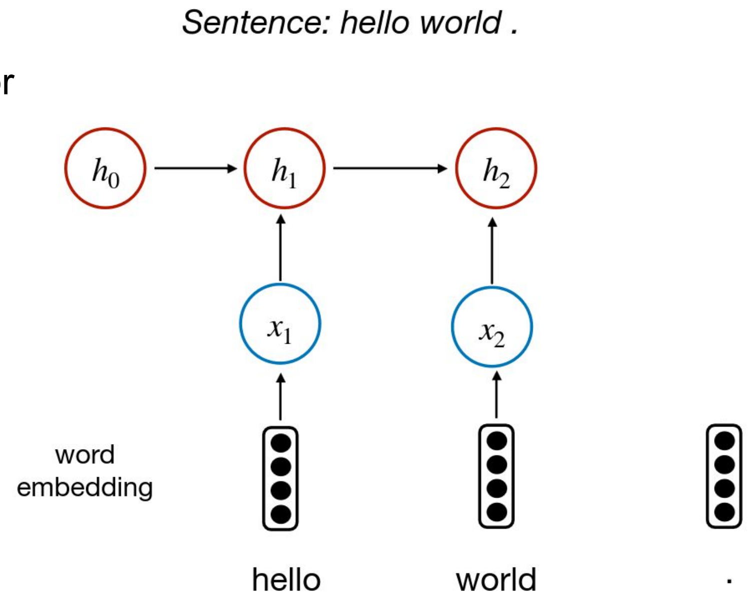
Step 2: Compute hidden state using word embedding and last hidden state



seq2seq encoder

Encoder: Transform some source sequence into a hidden representation

Step 1: Transform word to a vector (using embeddings matrix)
Step 2: Compute hidden state using word embedding and last hidden state

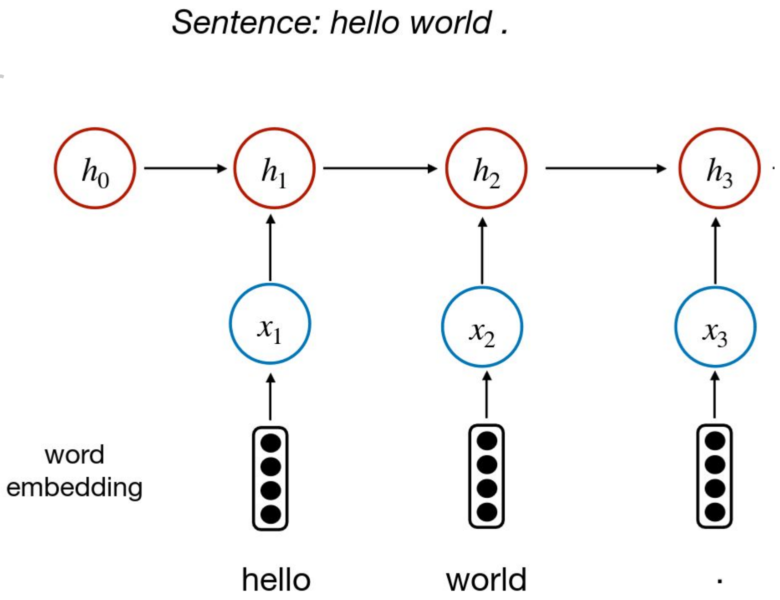


seq2seq encoder

Encoder: Transform some source sequence into a hidden representation

Step 1: Transform word to a vector (using embeddings matrix)

Step 2: Compute hidden state using word embedding and last hidden state

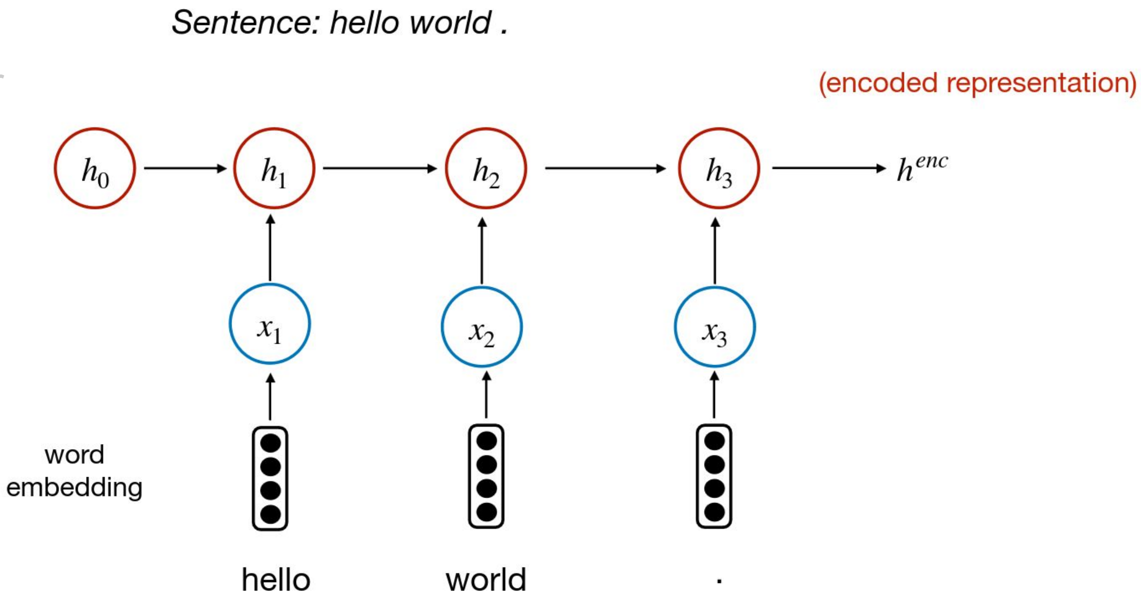


seq2seq encoder

Encoder: Transform some source sequence into a hidden representation

Step 1: Transform word to a vector (using embeddings matrix)
Step 2: Compute hidden state using word embedding and last hidden state

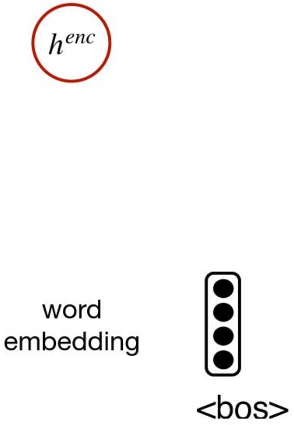
Key Idea: We've converted a *variable length* sequence to a *fixed length* representation



seq2seq decoder

Decoder: Using an encoded representation, predict a target sequence

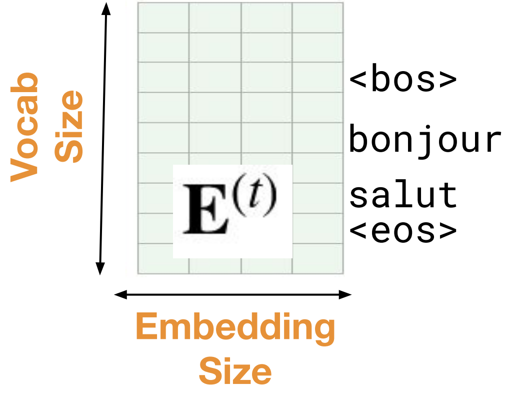
Step 1: Transform previous predicted token to word embedding

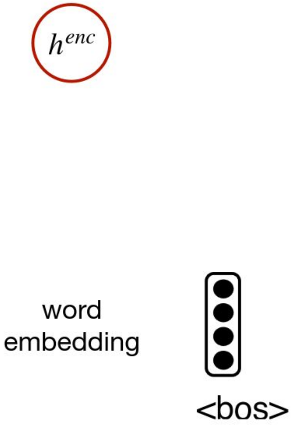


seq2seq decoder

Decoder: Using an encoded representation, predict a target sequence

Step 1: Transform previous predicted token to word embedding (using matrix $\mathbf{E}^{(t)}$)

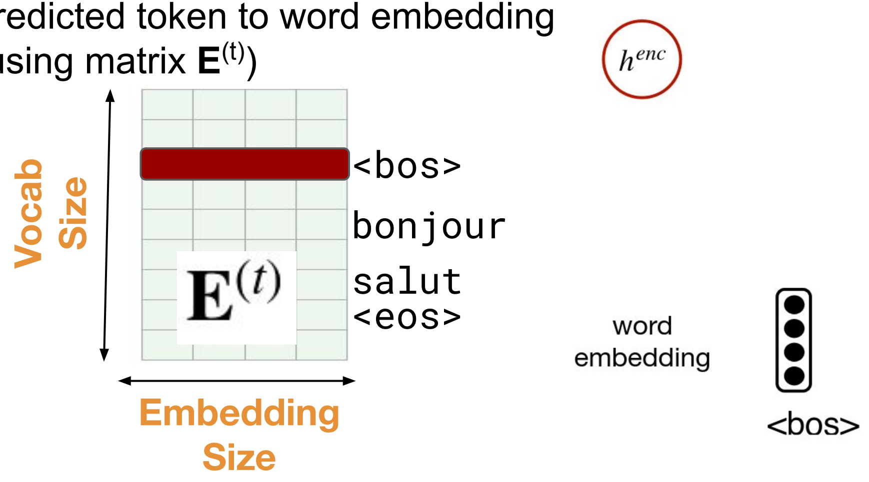




seq2seq decoder

Decoder: Using an encoded representation, predict a target sequence

Step 1: Transform previous predicted token to word embedding (using matrix $\mathbf{E}^{(t)}$)

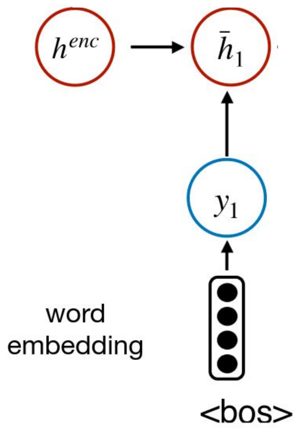


seq2seq decoder

Decoder: Using an encoded representation, predict a target sequence

Step 1: Transform previous predicted token to word embedding

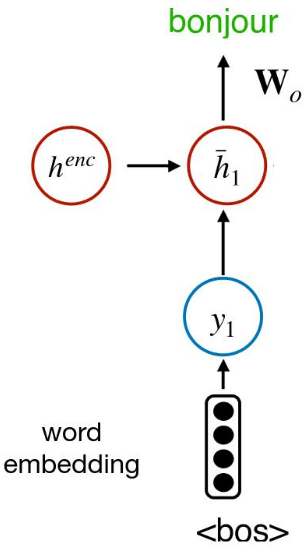
Step 2: Compute hidden state using word embedding and last hidden state



seq2seq decoder

Decoder: Using an encoded representation, predict a target sequence

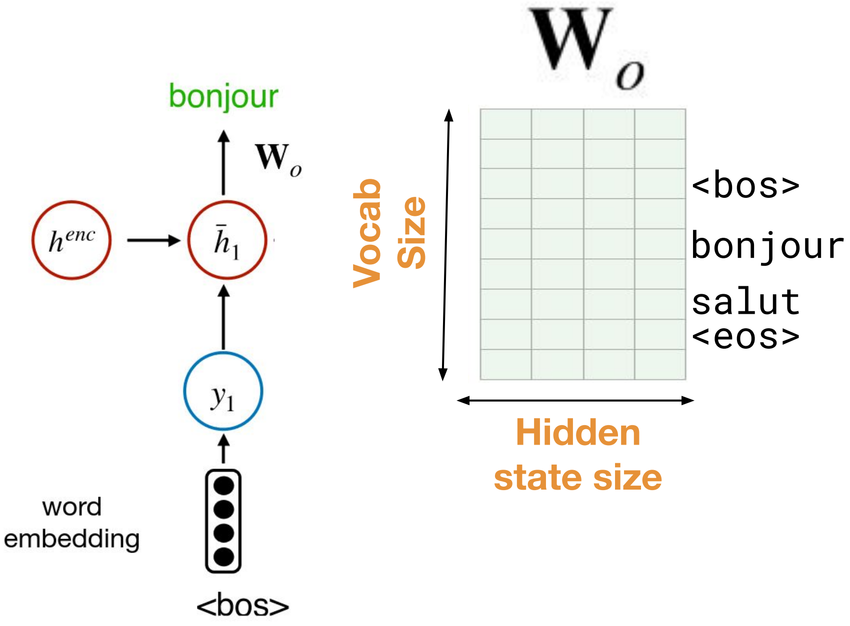
Step 1: Transform previous predicted token to word embedding
Step 2: Compute hidden state using word embedding and last hidden state
Step 3: Predict word using hidden state



seq2seq decoder

Decoder: Using an encoded representation, predict a target sequence

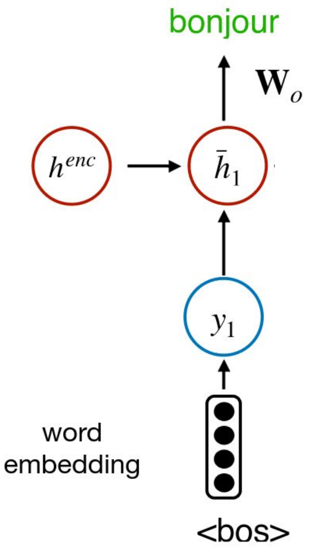
Step 1: Transform previous predicted token to word embedding
Step 2: Compute hidden state using word embedding and last hidden state
Step 3: Predict word using hidden state

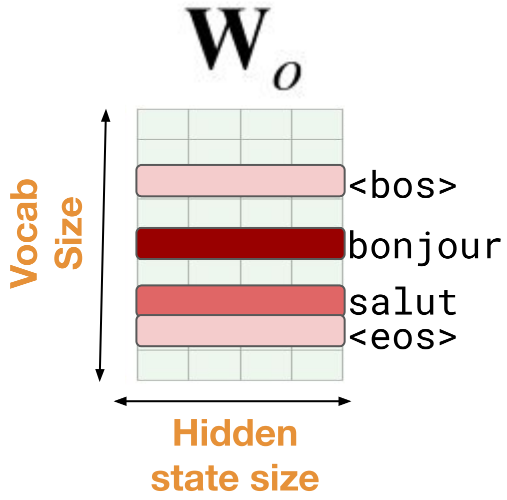


seq2seq decoder

Decoder: Using an encoded representation, predict a target sequence

Step 1: Transform previous predicted token to word embedding
Step 2: Compute hidden state using word embedding and last hidden state
Step 3: Predict word using hidden state





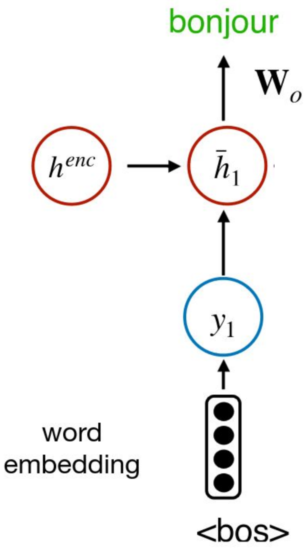
Recall: Output embeddings give us a **probability distribution** over outputs

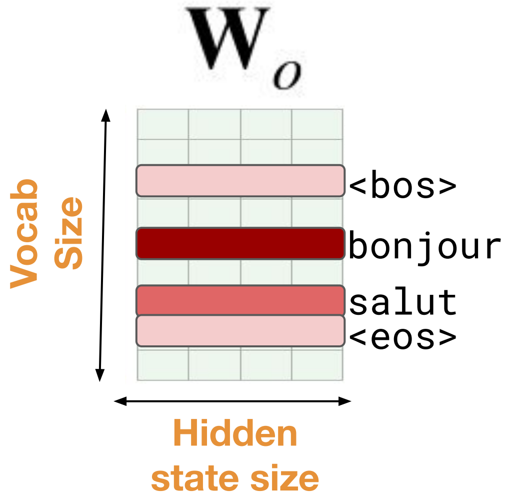
$$\mathbf{y}_t = \text{softmax}(\mathbf{h}_t W_O^\top)$$

seq2seq decoder

Decoder: Using an encoded representation, predict a target sequence

Step 1: Transform previous predicted token to word embedding
Step 2: Compute hidden state using word embedding and last hidden state
Step 3: Predict word using hidden state





$$\mathbf{y}_t = \mathrm{softmax}(\mathbf{h}_t W_O^\top)$$

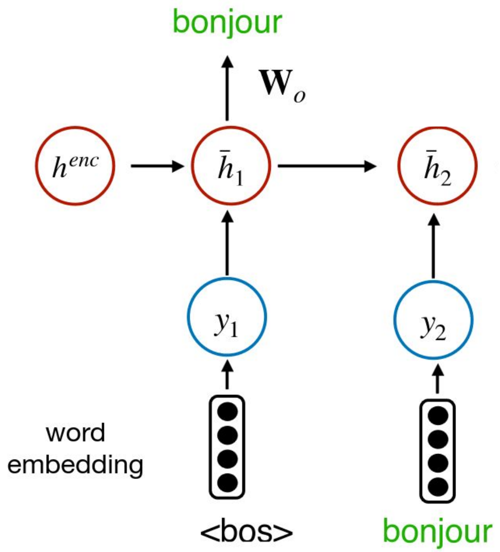
Recall: Output embeddings give us a **probability distribution** over outputs

Picking only the highest probability is called "greedy" decoding

seq2seq decoder

Decoder: Using an encoded representation, predict a target sequence

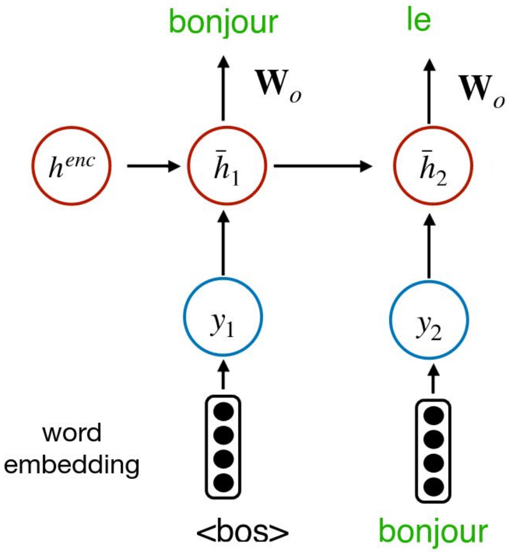
Step 1: Transform previous predicted token to word embedding
Step 2: Compute hidden state using word embedding and last hidden state
Step 3: Predict word using hidden state



seq2seq decoder

Decoder: Using an encoded representation, predict a target sequence

Step 1: Transform previous predicted token to word embedding
Step 2: Compute hidden state using word embedding and last hidden state
Step 3: Predict word using hidden state

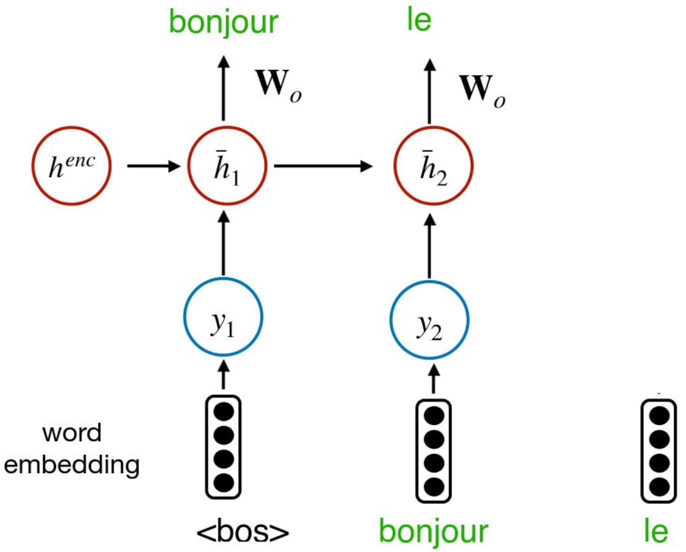


seq2seq decoder

Decoder: Using an encoded representation, predict a target sequence

Step 1: Transform previous predicted token to word embedding
Step 2: Compute hidden state using word embedding and last hidden state
Step 3: Predict word using hidden state

Repeat!



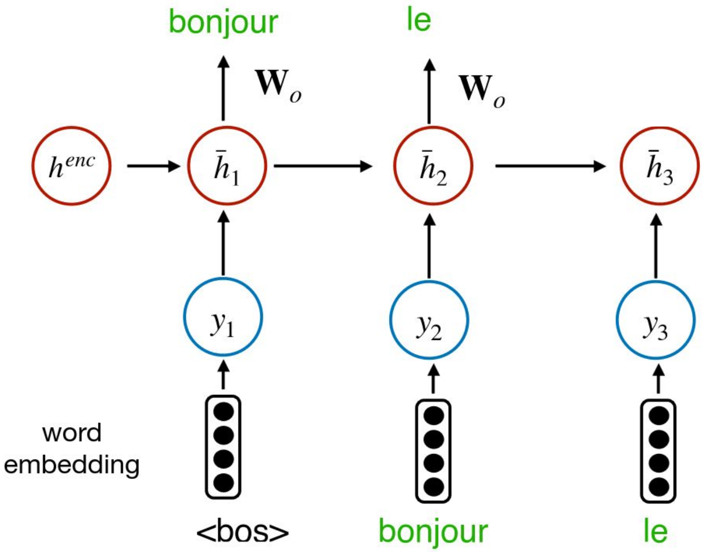
seq2seq decoder

Decoder: Using an encoded representation, predict a target sequence

Step 1: Transform previous predicted token to word embedding

Step 2: Compute hidden state using word embedding and last hidden state

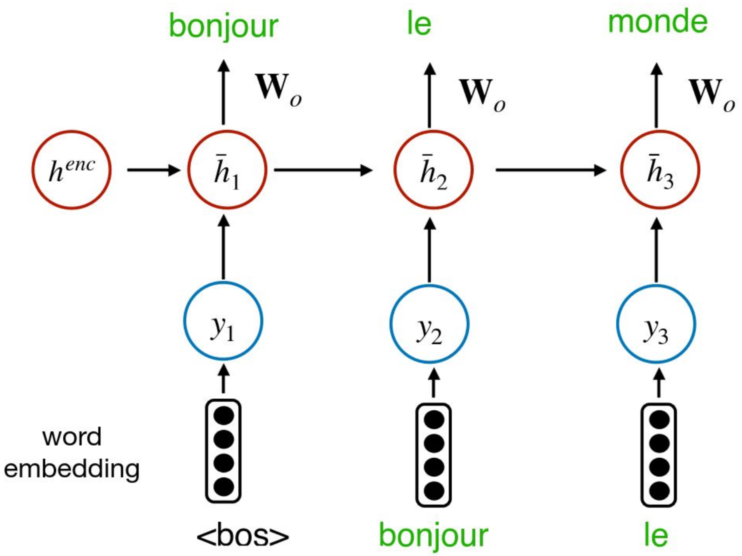
Step 3: Predict word using hidden state



seq2seq decoder

Decoder: Using an encoded representation, predict a target sequence

Step 1: Transform previous predicted token to word embedding
Step 2: Compute hidden state using word embedding and last hidden state
Step 3: Predict word using hidden state

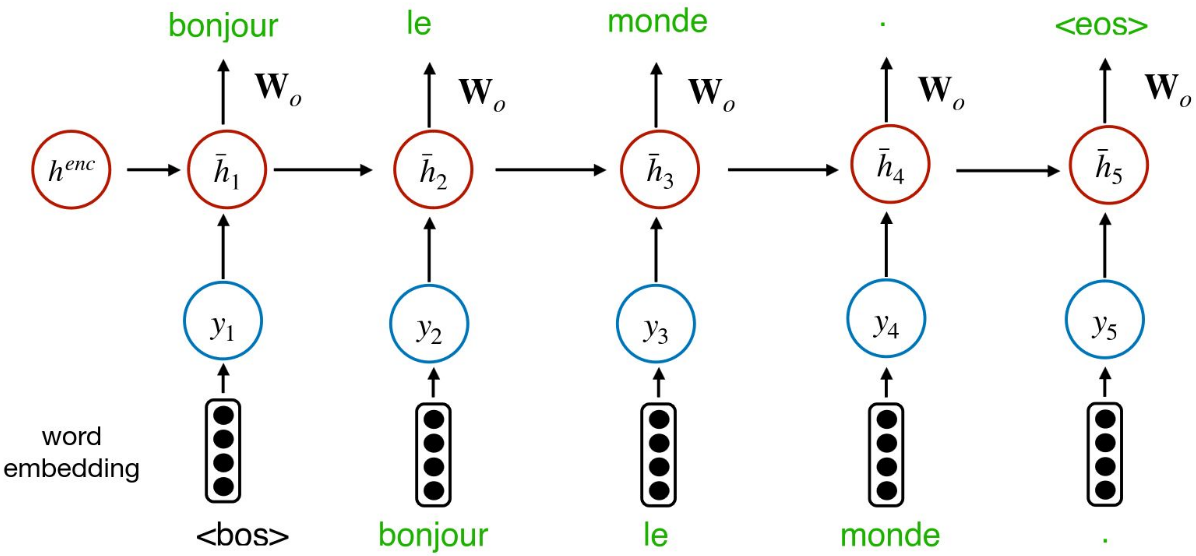


seq2seq decoder

Decoder: Using an encoded representation, predict a target sequence

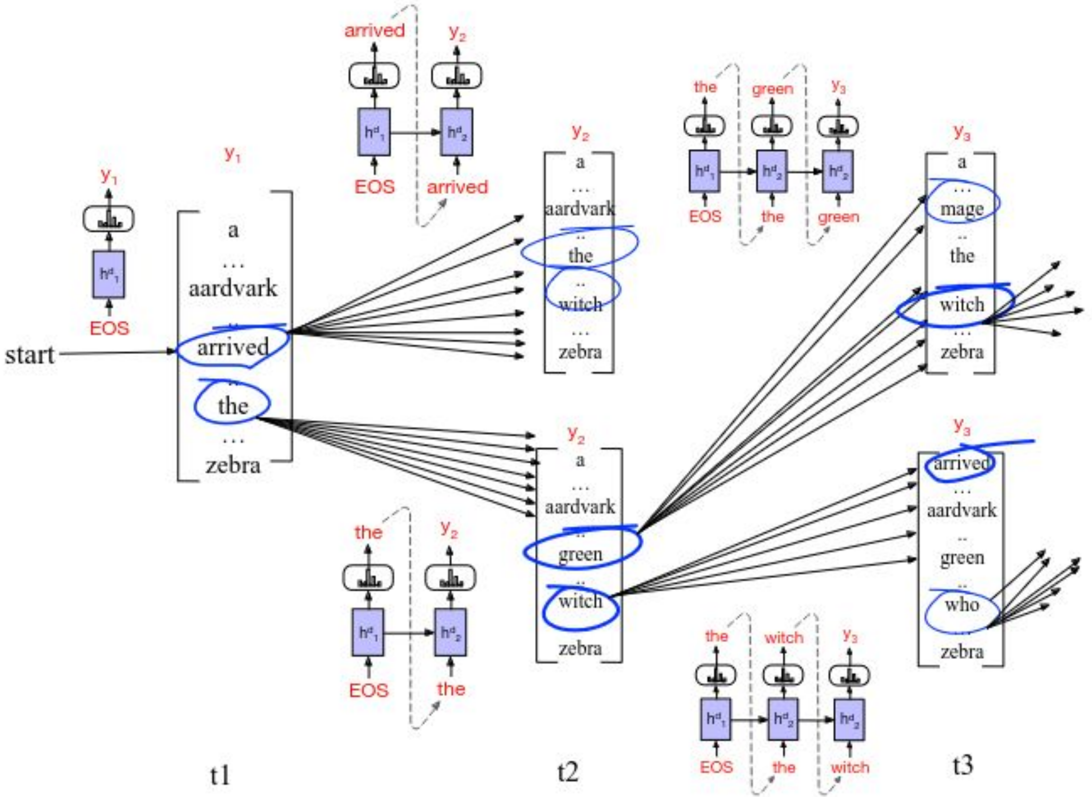
Step 1: Transform previous predicted token to word embedding
Step 2: Compute hidden state using word embedding and last hidden state
Step 3: Predict word using hidden state

Repeat process until model predicts <eos>



seq2seq Beam Search decoding

Key idea: Improve quality and variety of generations by tracking *k* best hypotheses at each step

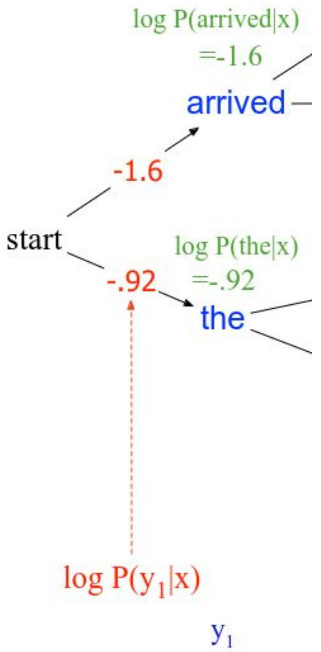


Beam Search Decoding Example (k=2)

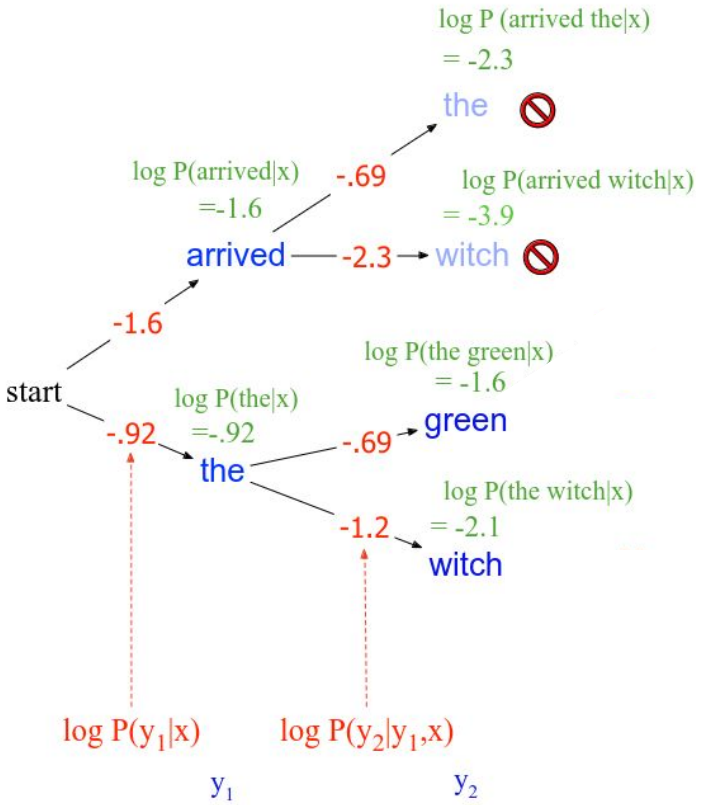
start

1

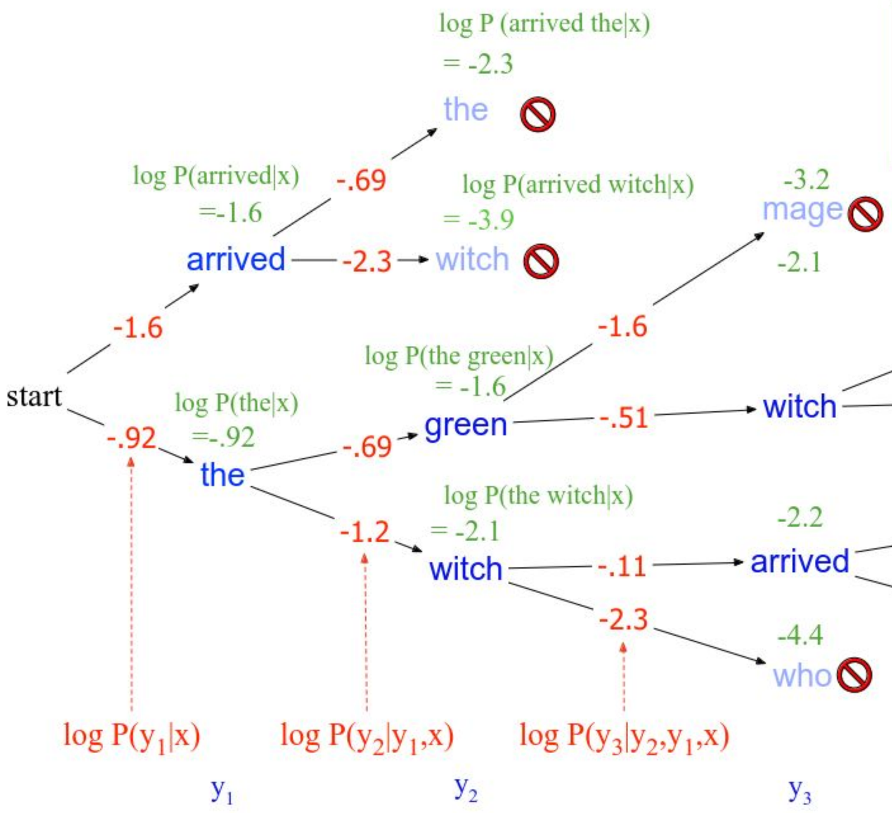
Beam Search Decoding Example (k=2)



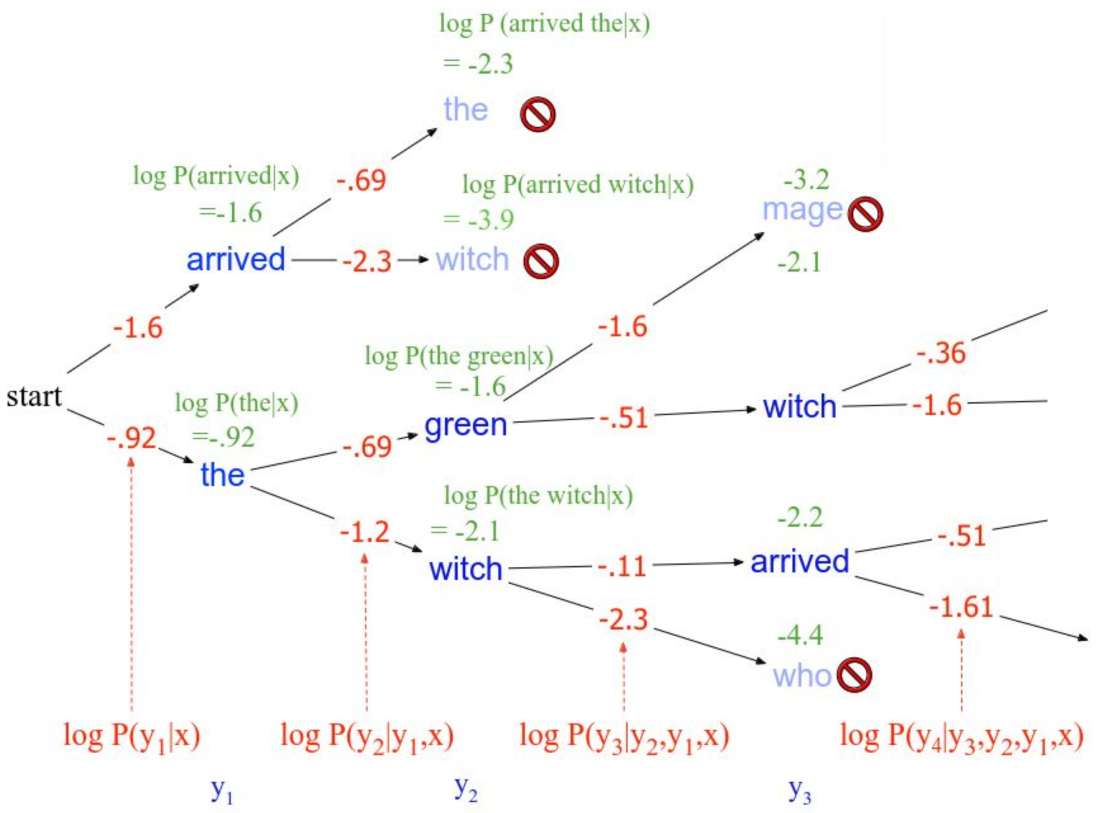
Beam Search Decoding Example (k=2)



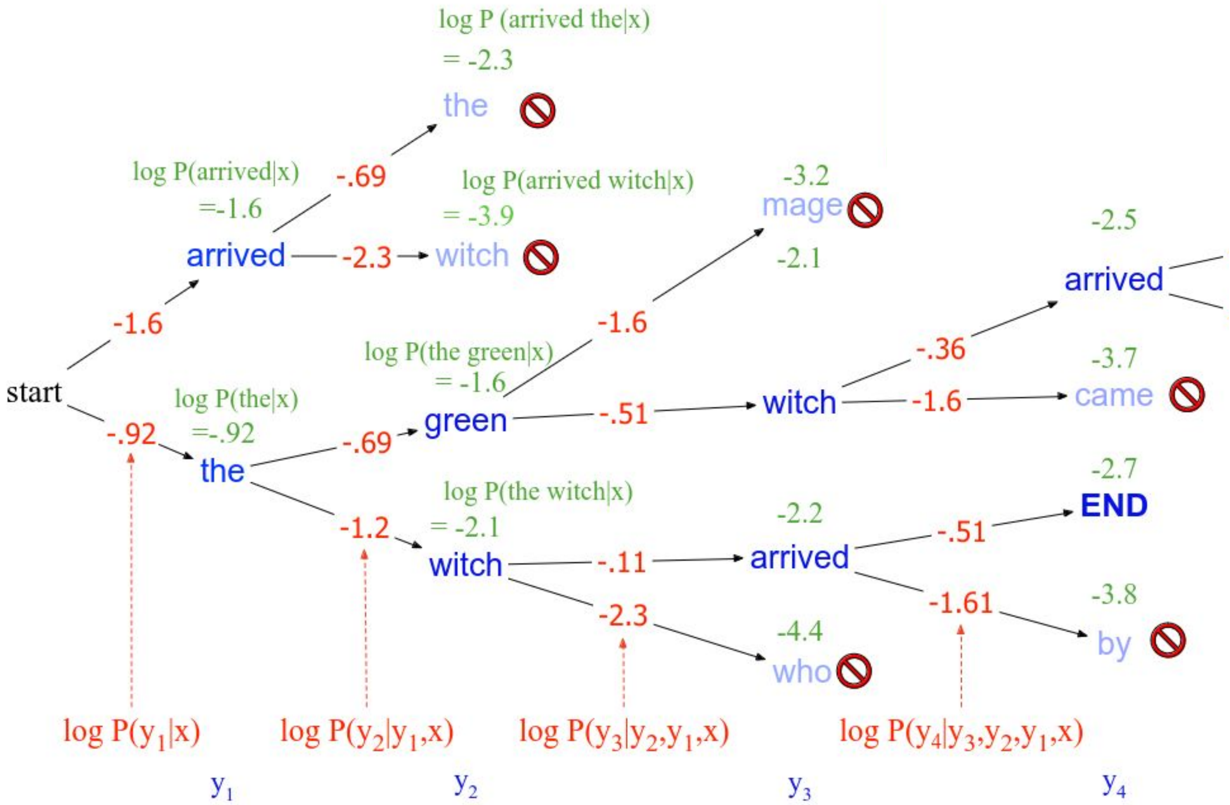
Beam Search Decoding Example (*k*=2)



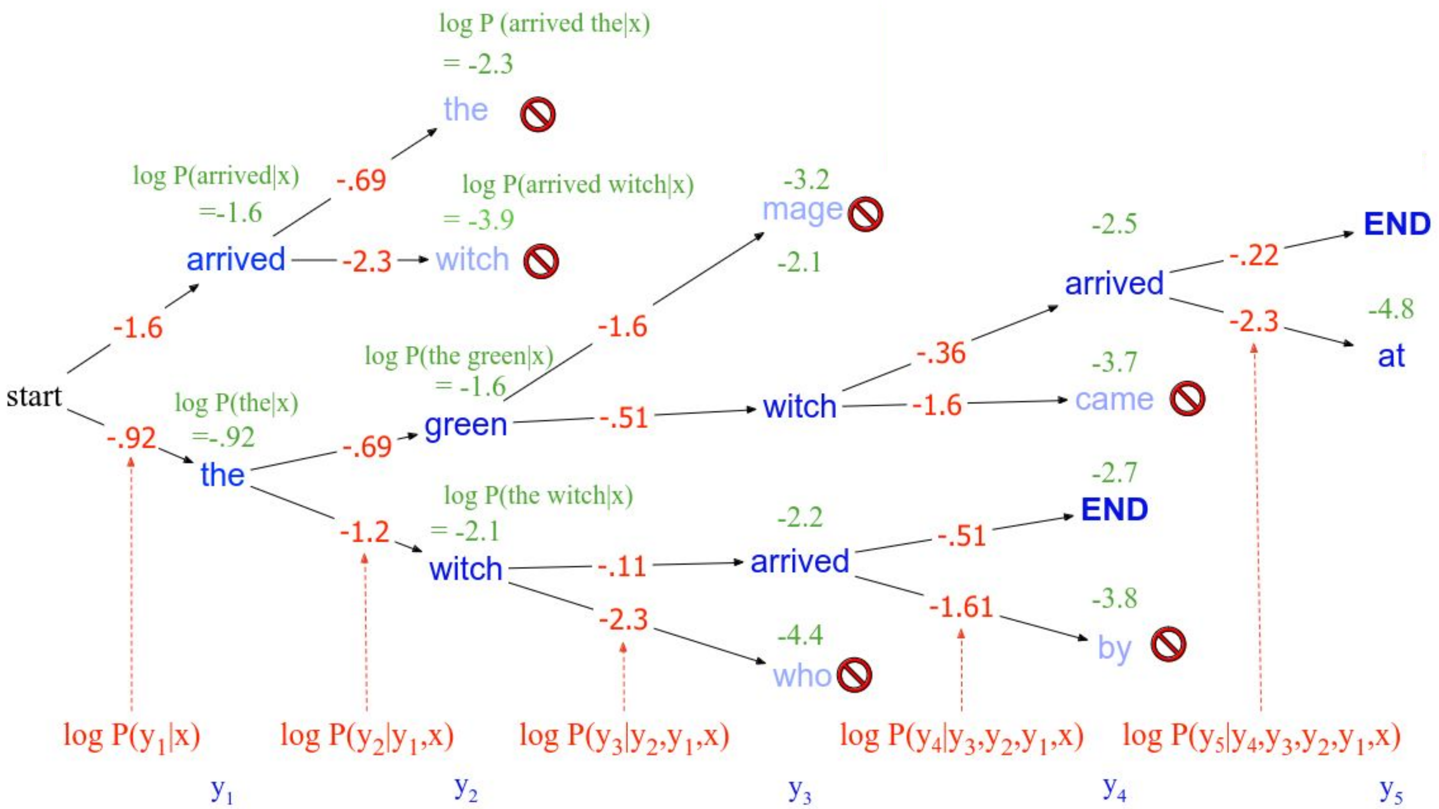
Beam Search Decoding Example (*k*=2)



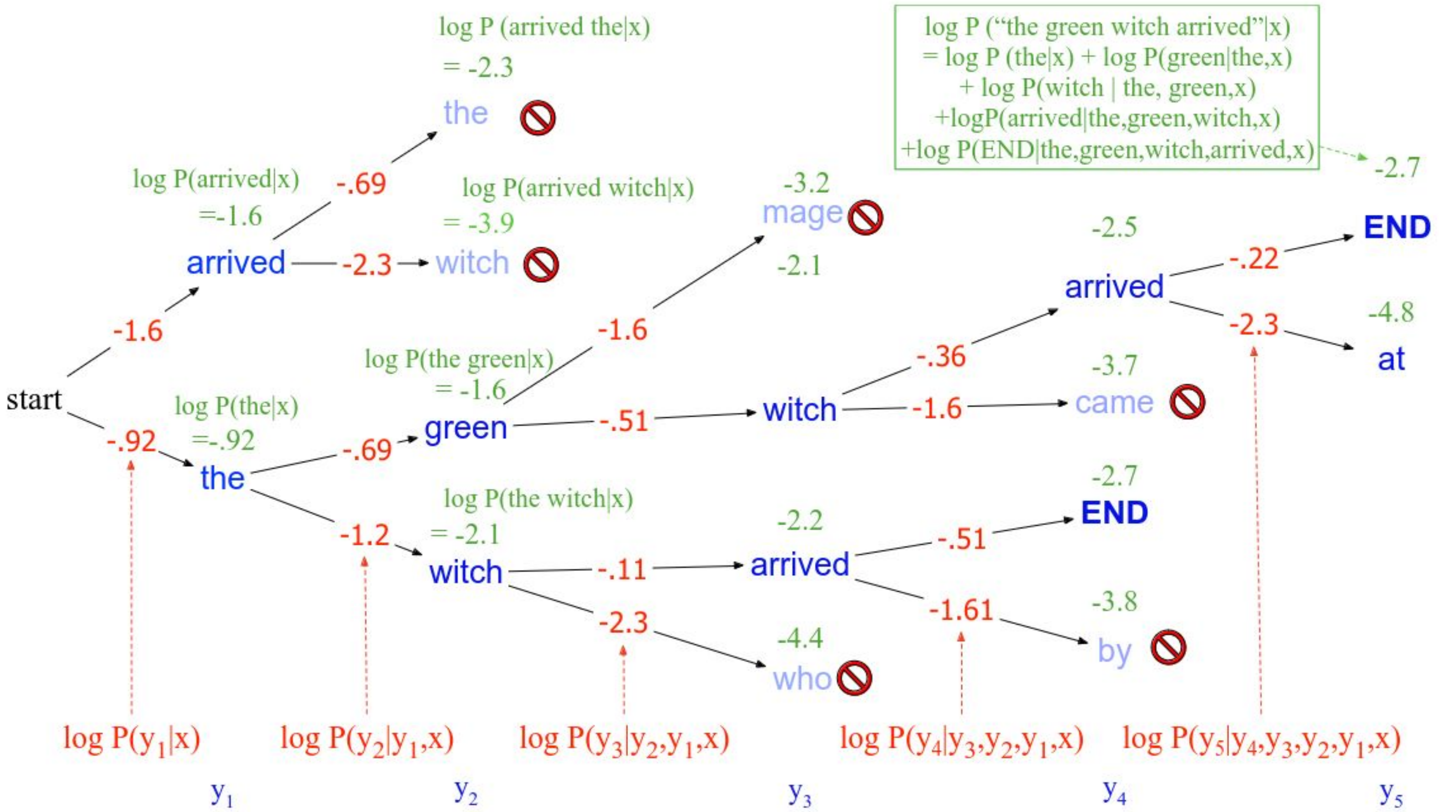
Beam Search Decoding Example (k=2)



Beam Search Decoding Example (k=2)

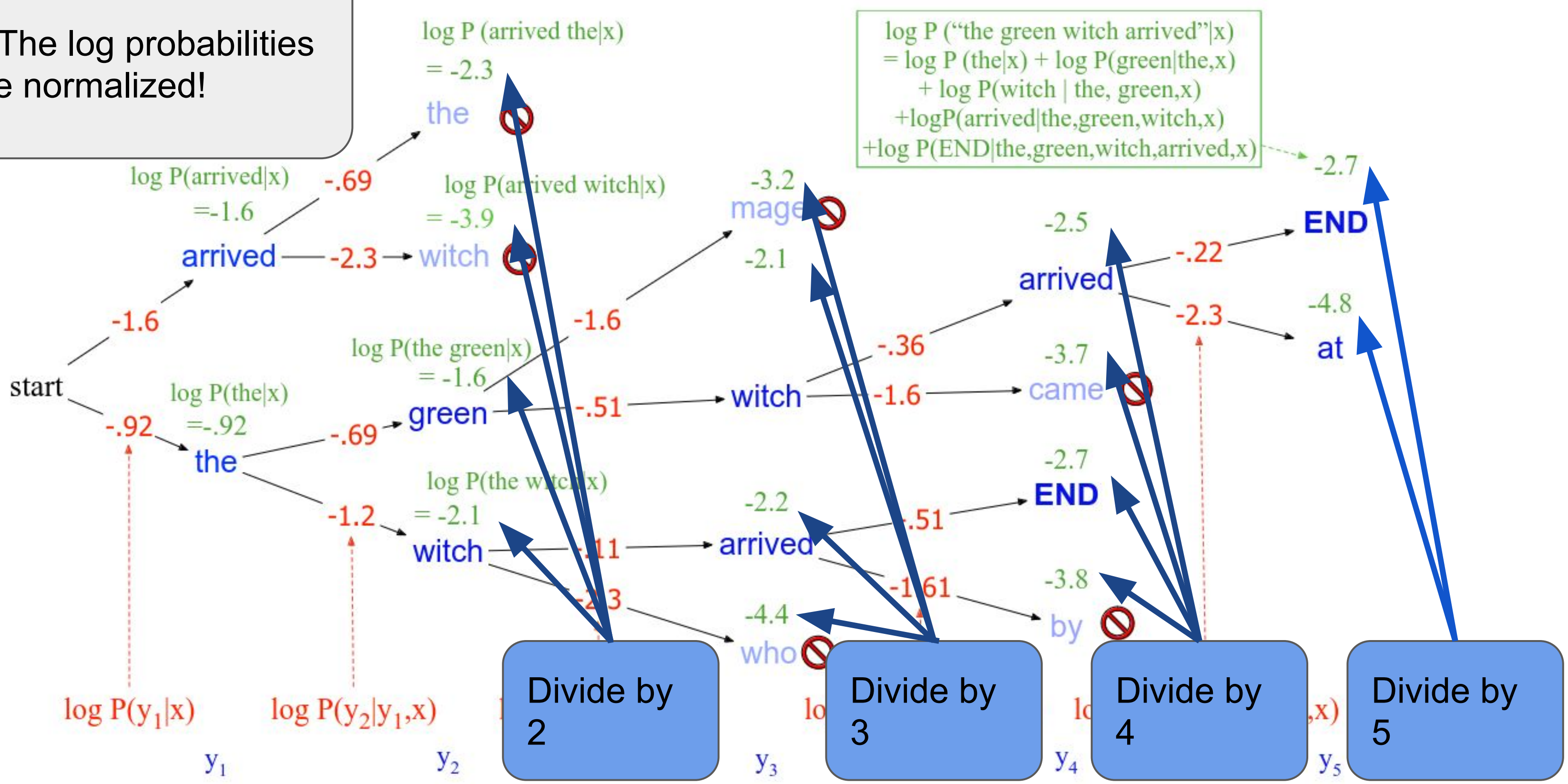


Beam Search Decoding Example (k=2)

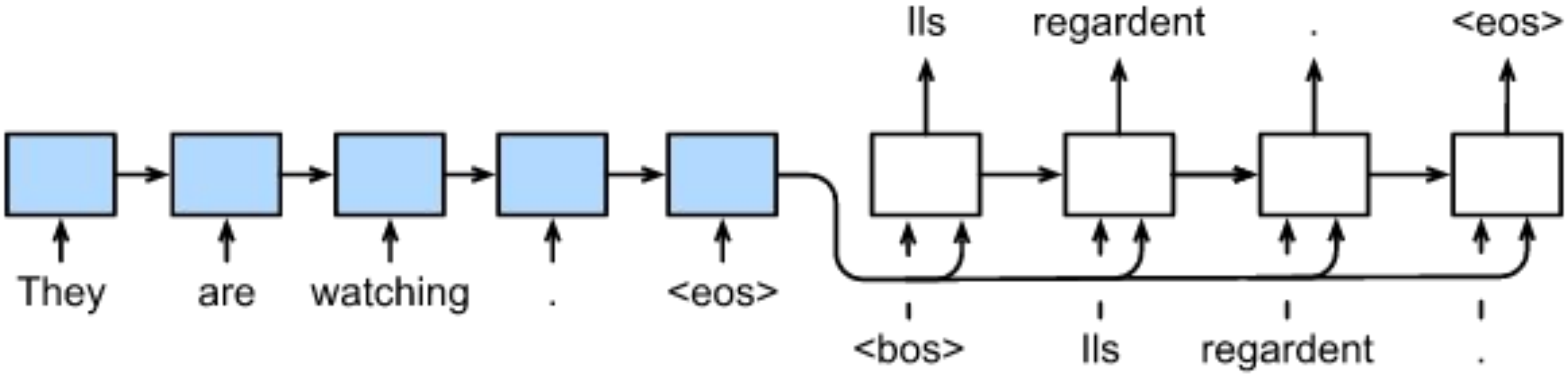


Beam Search Decoding Example (*k*=2)

Caveat: The log probabilities should be normalized!



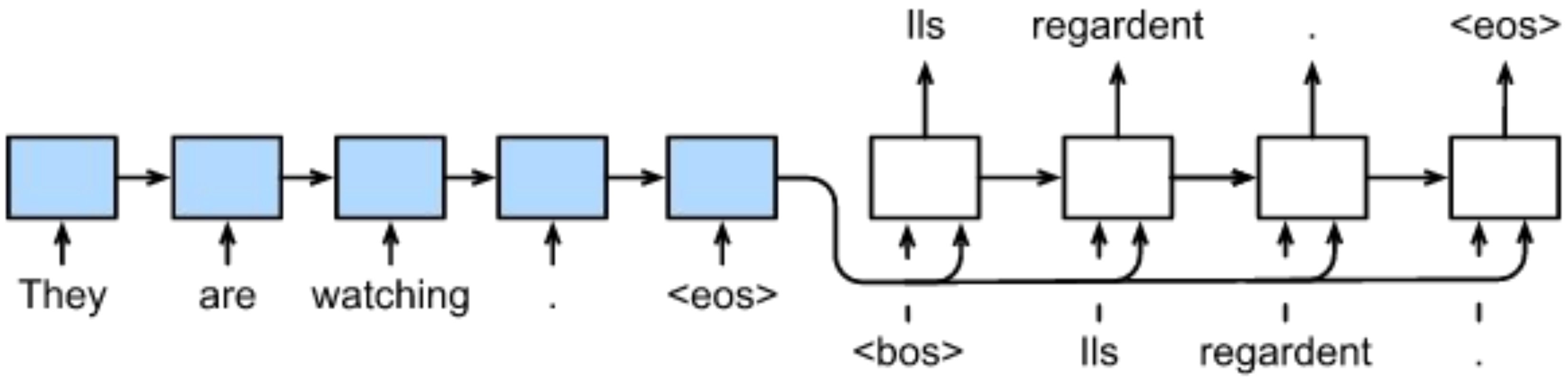
seq2seq



Vanilla seq2seq models are limited!

- Encoded representation is a “bottleneck” (must contain all relevant information from context!)
- Suffers from same issues as RNNs:
  - Vanishing gradients
  - Inefficient utilization of hardware

seq2seq



Vanilla seq2seq models are limited!

- Encoded representation is a “bottleneck” (must contain all relevant information from context!)
- Suffers from same issues as RNNs:
  - Vanishing gradients
  - Inefficient utilization of hardware

Adding attention to seq2seq can help solve representation bottleneck

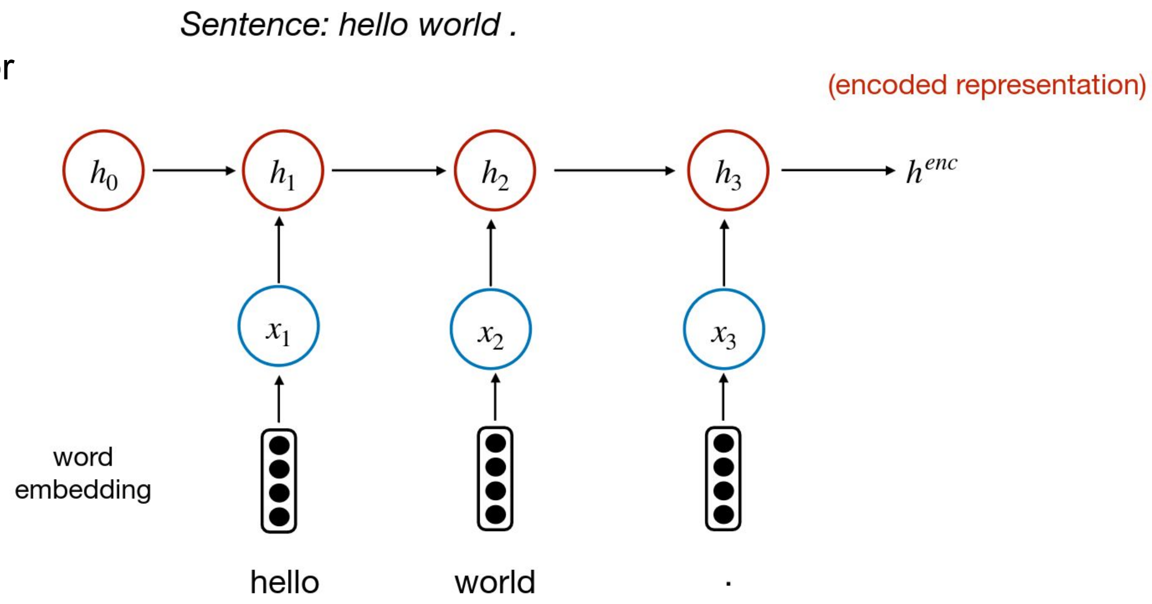
seq2seq with Attention

Key Idea: Let the decoder pick the parts of the encoder hidden states that it needs (i.e. “pay attention to” specific encoder hidden states)

seq2seq encoder with Attention

Encoder (with attention): Exactly the same as before! (except we also use hidden states h_1, h_2, h_3)

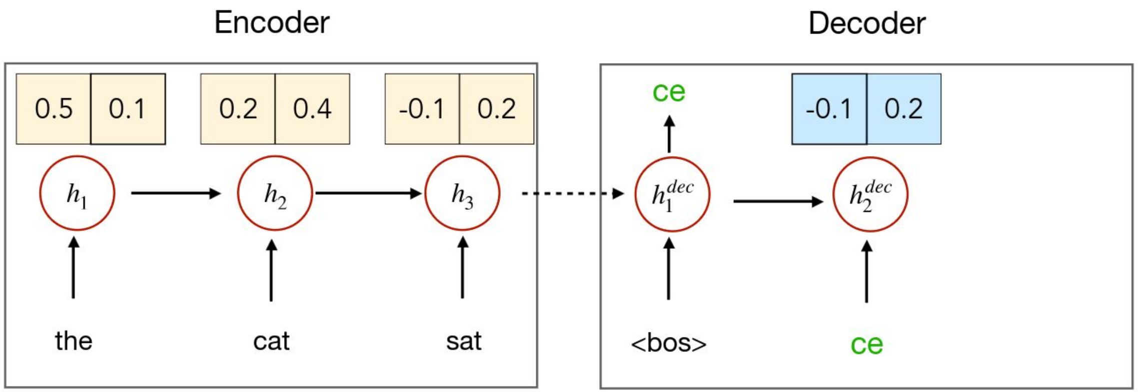
Step 1: Transform word to a vector (using embeddings matrix)
Step 2: Compute hidden state using word embedding and last hidden state



seq2seq decoder with Attention

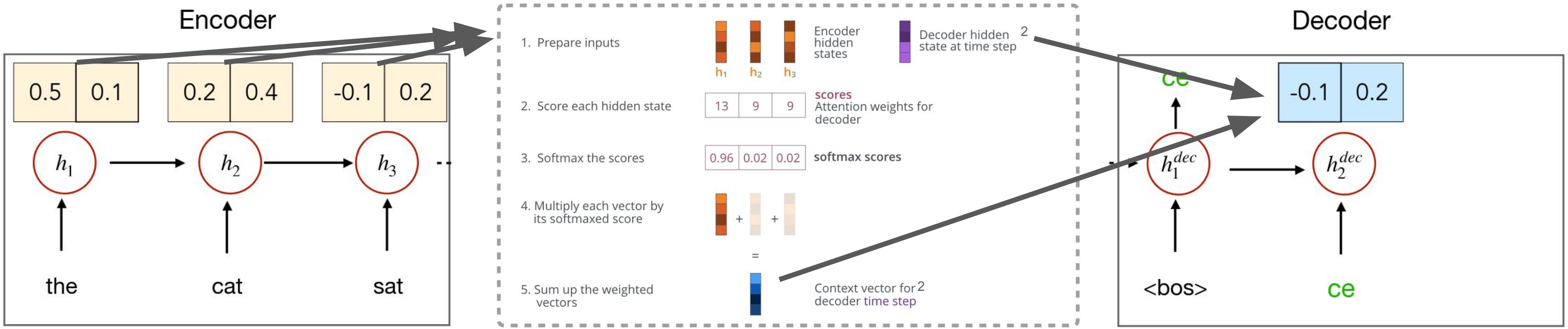
Decoder (with Attention): Using **all hidden states from the encoder**, predict a target sequence

Step 1: Transform previous predicted token to word embedding
Step 2: Compute decoder hidden state using word embedding
Step 3 (new): Compute context for decoder using all encoder hidden states
Step 4: Predict word using hidden state **combined with context vector**



seq2seq decoder with Attention

Decoder (with Attention): Using **all hidden states from the encoder**, predict a target sequence

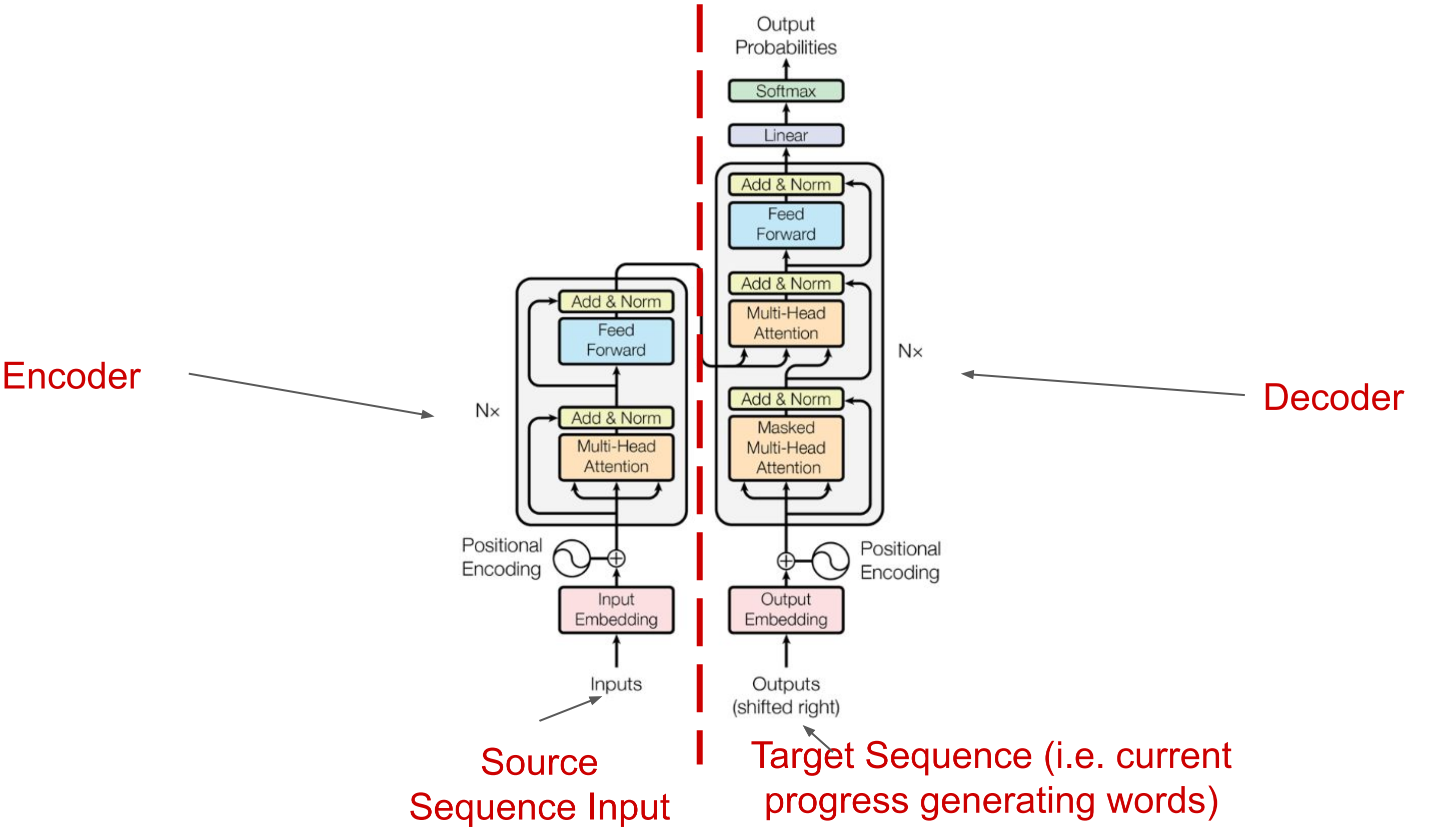


Attention: Mathematical formulation

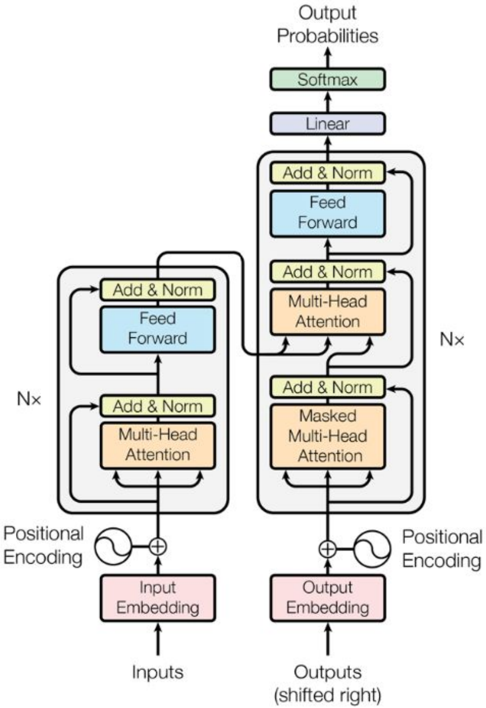
- Encoder hidden states: $h_1^{enc}, \ldots, h_n^{enc}$
  (n: # of words in source sentence)
- Decoder hidden state at time t: h_t^{dec}
- Attention scores:
  $$e^t = [g(h_1^{enc}, h_t^{dec}), \ldots, g(h_n^{enc}, h_t^{dec})] \in \mathbb{R}^n$$
- Attention distribution:
  $$\alpha^t = \text{softmax}(e^t) \in \mathbb{R}^n$$
- Weighted sum of encoder hidden states:
  $$a_t = \sum_{i=1}^{n} \alpha_i^t h_i^{enc} \in \mathbb{R}^h$$

Combine a_t and h_t^{dec} to predict next word

Transformer Architecture



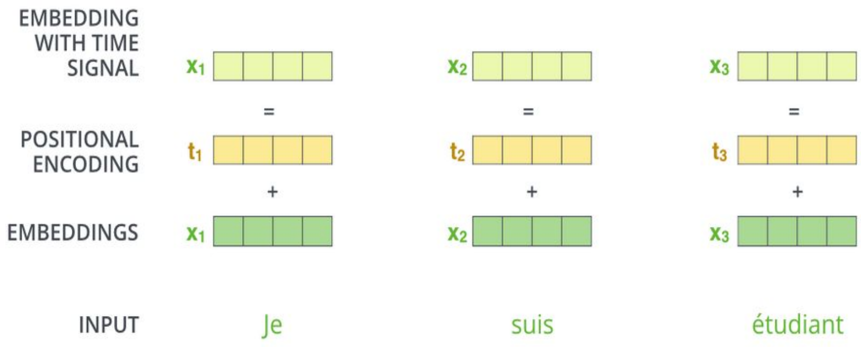
Transformer Encoder



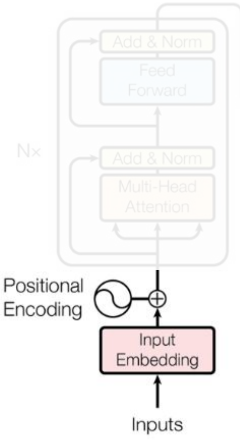
Source sequence $(x_1, \ldots, x_n)$

Transformer Encoder: Positional + Word Embedding

Input and Positional Embedding



Embedded source sequence
$\mathbb{R}^{n \times d_1}$



Source sequence
$(x_1, \ldots, x_n)$

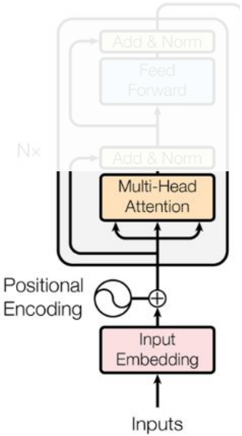
Transformer Encoder: Multi-Head Self Attention

After Multi-Head Attention

$\mathbb{R}^{n \times d_2}$

Embedded source sequence

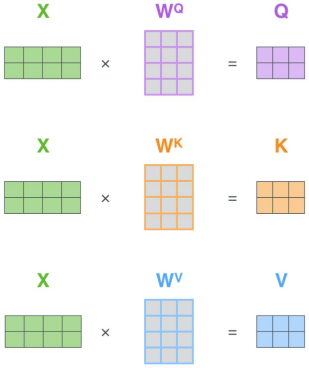
$\mathbb{R}^{n \times d_1}$



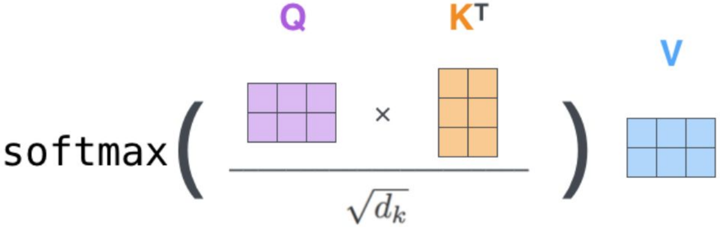
Source sequence $(x_1, \ldots, x_n)$

Self-Attention: $W_i^Q \in \mathbb{R}^{d_1 \times d_q}, W_i^K \in \mathbb{R}^{d_1 \times d_k}, W_i^V \in \mathbb{R}^{d_1 \times d_v}$

Step 1:



Step 2:



MultiHead Attention: $W^O \in \mathbb{R}^{d \times d_2}$

$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head}_1, \ldots, \mathrm{head}_\mathrm{h})W^O$$

$$\mathrm{head}_i = \mathrm{Attention}(XW_i^Q, XW_i^K, XW_i^V)$$

Transformer Encoder: Multi-Head Self Attention

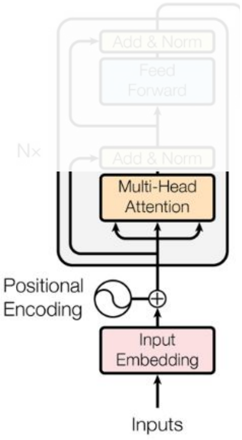
In practice, $d_1 = d_2$

After Multi-Head Attention

$\mathbb{R}^{n \times d_1}$

Embedded source sequence

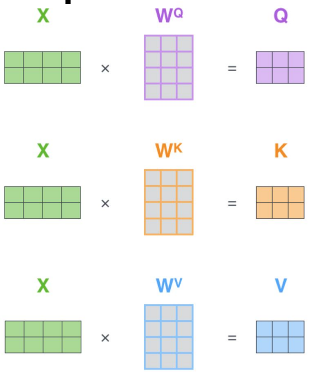
$\mathbb{R}^{n \times d_1}$



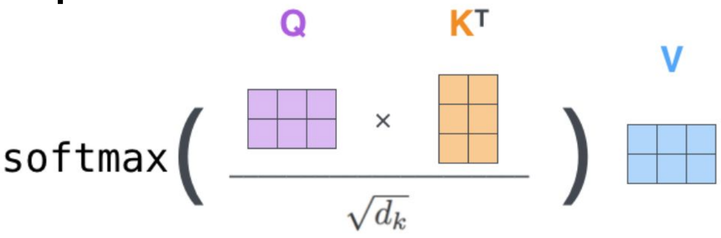
Source sequence $(x_1, \ldots, x_n)$

Self-Attention: $W_i^Q \in \mathbb{R}^{d_1 \times d_q}, W_i^K \in \mathbb{R}^{d_1 \times d_k}, W_i^V \in \mathbb{R}^{d_1 \times d_v}$

Step 1:



Step 2:



"Self" attention means Q, K, V are all computed from a single sequence

MultiHead Attention: $W^O \in \mathbb{R}^{d \times d_2}$

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_\text{h})W^O$$

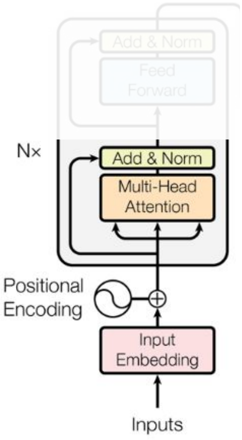
$$\text{head}_i = \text{Attention}(XW_i^Q, XW_i^K, XW_i^V)$$

Transformer Encoder: Add & Norm

After Add & Norm
$\mathbb{R}^{n\times d_1}$

After Multi-Head Attention
$\mathbb{R}^{n\times d_1}$

Embedded source sequence
$\mathbb{R}^{n\times d_1}$



Source sequence
$(x_1, \ldots, x_n)$

Add & Norm:

$$\mathrm{LayerNorm}(x + \mathrm{Sublayer}(x))$$

LayerNorm

$$\boxed{y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta}$$

Transformer Encoder: Feed Forward

After Feed Forward

$\mathbb{R}^{n \times d_1}$

After Add & Norm

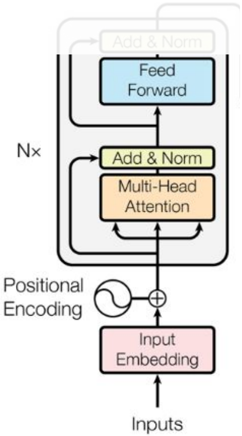
$\mathbb{R}^{n \times d_1}$

After Multi-Head Attention

$\mathbb{R}^{n \times d_1}$

Embedded source sequence

$\mathbb{R}^{n \times d_1}$



Source sequence $(x_1, \ldots, x_n)$

Feed Forward

$$\mathrm{FFN}(\mathbf{x}_i) = \mathrm{ReLU}(\mathbf{x}_i \mathbf{W}_1 + \mathbf{b}_1)\mathbf{W_2} + \mathbf{b}_2$$

$$\mathbf{W}_1 \in \mathbb{R}^{d \times d_{ff}}, \mathbf{b}_1 \in \mathbb{R}^{d_{ff}}$$

$$\mathbf{W}_2 \in \mathbb{R}^{d_{ff} \times d}, \mathbf{b}_2 \in \mathbb{R}^{d}$$

Compute transformation over each value in the sequence **independently**

Transformer Encoder: Final Add & Norm

After Final Add & Norm

$\mathbb{R}^{n \times d_1}$

After Feed Forward

$\mathbb{R}^{n \times d_1}$

After Add & Norm

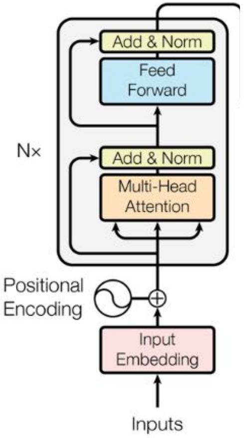
$\mathbb{R}^{n \times d_1}$

After Multi-Head Attention

$\mathbb{R}^{n \times d_1}$

Embedded source sequence

$\mathbb{R}^{n \times d_1}$



Source sequence (x_1, …, x_n)

Add & Norm:

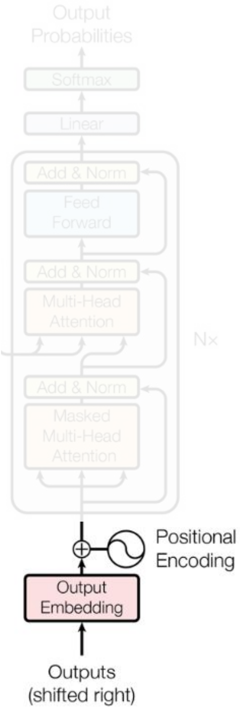
$$\mathrm{LayerNorm}(x + \mathrm{Sublayer}(x))$$

LayerNorm

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$

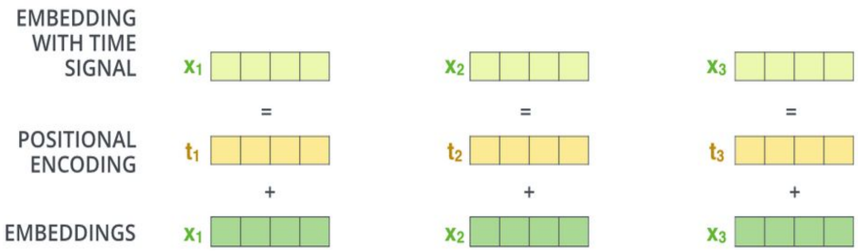
Transformer Decoder:

Embedded target sequence
$\mathbb{R}^{m \times d_1}$



Target sequence
(<bos>, x_1, …, x_m)

Output and Positional Embedding



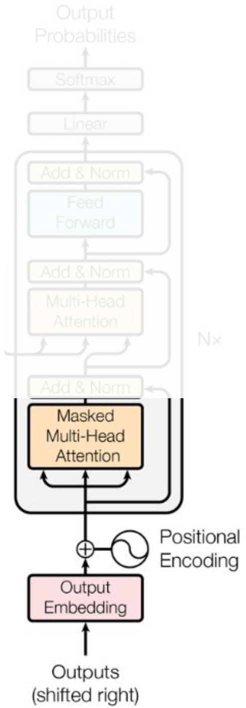
Transformer Decoder: Masked Multi-Head Attention

Masked Multi-Head Attention

$\mathbb{R}^{m \times d_1}$

Embedded target sequence

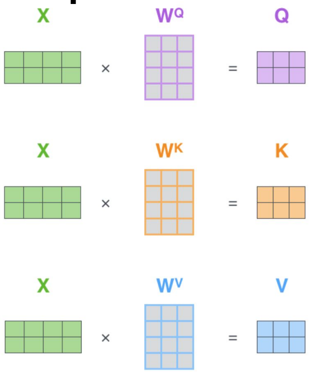
$\mathbb{R}^{m \times d_1}$



Target sequence
(<bos>, x_1, …, x_m)

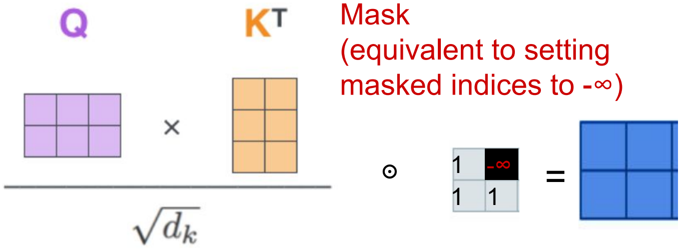
Masked Self-Attention: $W_i^Q \in \mathbb{R}^{d_1 \times d_q}, W_i^K \in \mathbb{R}^{d_1 \times d_k}, W_i^V \in \mathbb{R}^{d_1 \times d_v}$

Step 1:

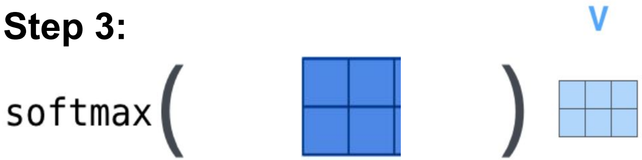


Step 2:

Elementwise Multiply by Mask
(equivalent to setting masked indices to -∞)



Step 3:



MultiHead Attention: $W^O \in \mathbb{R}^{d \times d_2}$

$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head}_1, \ldots, \mathrm{head_h})W^O$$

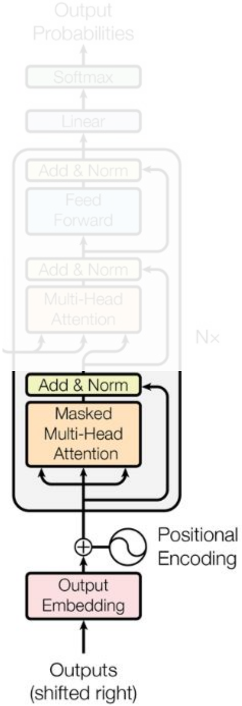
$$\mathrm{head}_i = \mathrm{Attention}(XW_i^Q, XW_i^K, XW_i^V)$$

Transformer Decoder:

After Add & Norm
$\mathbb{R}^{m \times d_1}$

Masked Multi-Head Attention
$\mathbb{R}^{m \times d_1}$

Embedded target sequence
$\mathbb{R}^{m \times d_1}$



Target sequence
(<bos>, x_1, …, x_m)

Add & Norm:

$$\text{LayerNorm}(x + \text{Sublayer}(x))$$

LayerNorm

$$\boxed{y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta}$$

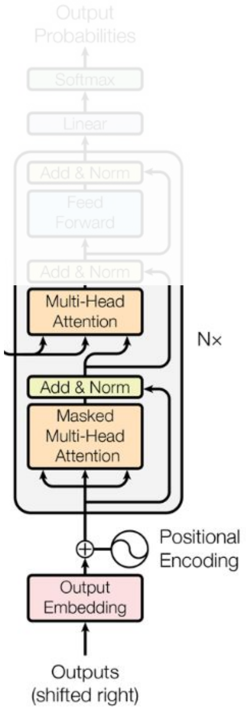
Transformer Decoder: Multi-Head (Cross) Attention

Masked Multi-Head Attention
$\mathbb{R}^{m \times d_1}$

After Add & Norm
$\mathbb{R}^{m \times d_1}$

Masked Multi-Head Attention
$\mathbb{R}^{m \times d_1}$

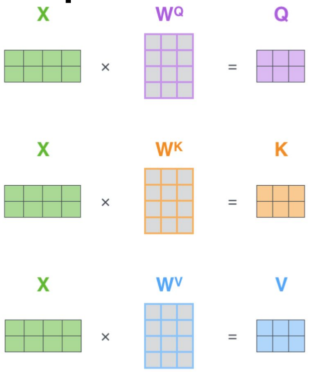
Embedded target sequence
$\mathbb{R}^{m \times d_1}$



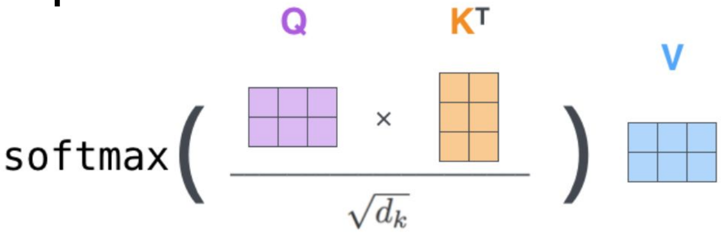
Target sequence
(<bos>, x_1, …, x_m)

Cross-Attention: $W_i^Q \in \mathbb{R}^{d_1 \times d_q}, W_i^K \in \mathbb{R}^{d_1 \times d_k}, W_i^V \in \mathbb{R}^{d_1 \times d_v}$

Step 1:



Step 2:



"Cross" attention means Q, K, V are computed from **separate** sequences

MultiHead Attention: $W^O \in \mathbb{R}^{d \times d_2}$

$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head}_1, \ldots, \mathrm{head}_h) W^O$$

$$\mathrm{head}_i = \mathrm{Attention}(XW_i^Q, XW_i^K, XW_i^V)$$

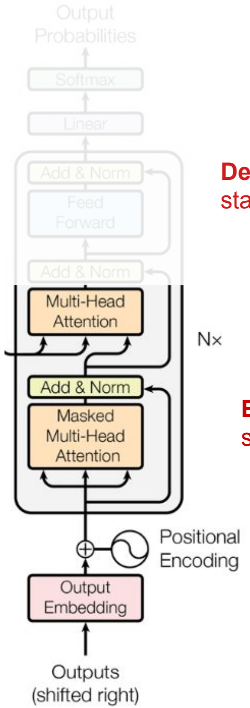
Transformer Decoder: Multi-Head (Cross) Attention

Masked Multi-Head Attention
$\mathbb{R}^{m\times d_1}$

After Add & Norm
$\mathbb{R}^{m\times d_1}$

Masked Multi-Head Attention
$\mathbb{R}^{m\times d_1}$

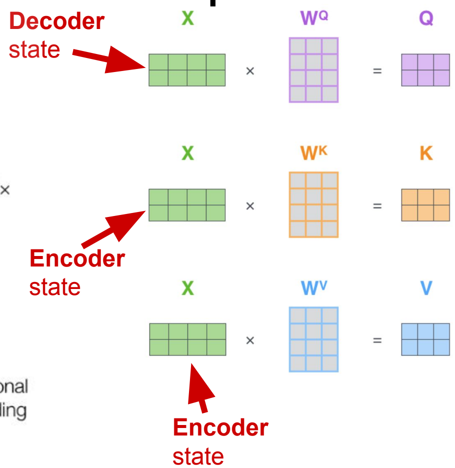
Embedded target sequence
$\mathbb{R}^{m\times d_1}$



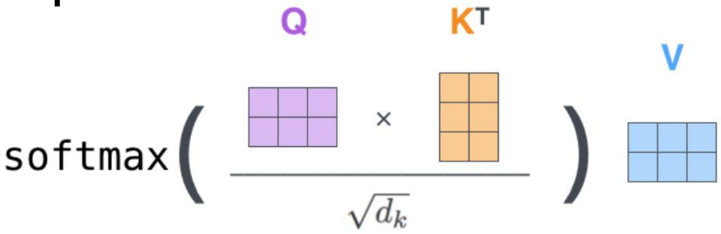
Target sequence
(<bos>, x_1, …, x_m)

Cross-Attention: $W_i^Q \in \mathbb{R}^{d_1\times d_q}, W_i^K \in \mathbb{R}^{d_1\times d_k}, W_i^V \in \mathbb{R}^{d_1\times d_v}$

Step 1:



Step 2:



"Cross" attention means Q, K, V are computed from **separate** sequences

MultiHead Attention: $W^O \in \mathbb{R}^{d\times d_2}$

$$\mathrm{MultiHead}(Q,K,V) = \mathrm{Concat}(\mathrm{head_1},\ldots,\mathrm{head_h})W^O$$

$$\mathrm{head_i} = \mathrm{Attention}(XW_i^Q, XW_i^K, XW_i^V)$$

Transformer Decoder: Add & Norm

Add & Norm

$\mathbb{R}^{m\times d_1}$

Masked Multi-Head Attention

$\mathbb{R}^{m\times d_1}$

After Add & Norm

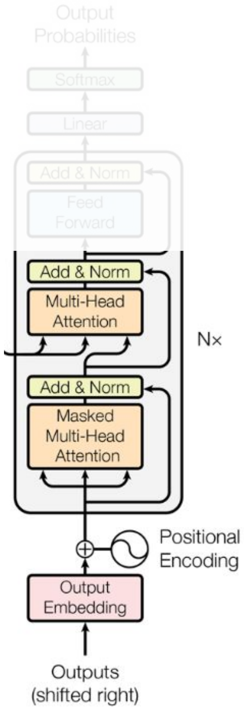
$\mathbb{R}^{m\times d_1}$

Masked Multi-Head Attention

$\mathbb{R}^{m\times d_1}$

Embedded target sequence

$\mathbb{R}^{m\times d_1}$



Target sequence
(<bos>, x_1, …, x_m)

Add & Norm:

$$\mathrm{LayerNorm}(x + \mathrm{Sublayer}(x))$$

LayerNorm

$$\boxed{y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta}$$

Transformer Decoder: Feed Forward

Feed Forward

$\mathbb{R}^{m \times d_1}$

Add & Norm

$\mathbb{R}^{m \times d_1}$

Masked Multi-Head Attention

$\mathbb{R}^{m \times d_1}$

After Add & Norm

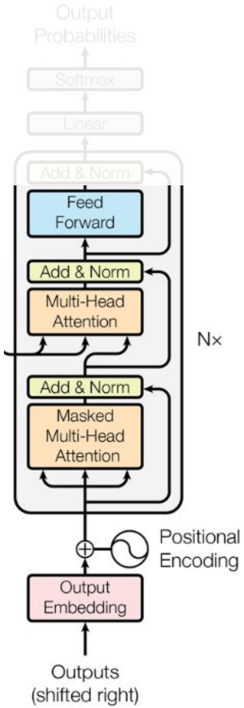
$\mathbb{R}^{m \times d_1}$

Masked Multi-Head Attention

$\mathbb{R}^{m \times d_1}$

Embedded target sequence

$\mathbb{R}^{m \times d_1}$



Target sequence
(<bos>, x_1, …, x_m)

Feed Forward

$$\mathrm{FFN}(\mathbf{x}_i) = \mathrm{ReLU}(\mathbf{x}_i \mathbf{W}_1 + \mathbf{b}_1)\mathbf{W_2} + \mathbf{b}_2$$

$$\mathbf{W}_1 \in \mathbb{R}^{d \times d_{ff}}, \mathbf{b}_1 \in \mathbb{R}^{d_{ff}}$$

$$\mathbf{W}_2 \in \mathbb{R}^{d_{ff} \times d}, \mathbf{b}_2 \in \mathbb{R}^{d}$$

Transformer Decoder: Add & Norm

Add & Norm

$\mathbb{R}^{m \times d_1}$

Feed Forward

$\mathbb{R}^{m \times d_1}$

Add & Norm

$\mathbb{R}^{m \times d_1}$

Masked Multi-Head Attention

$\mathbb{R}^{m \times d_1}$

After Add & Norm

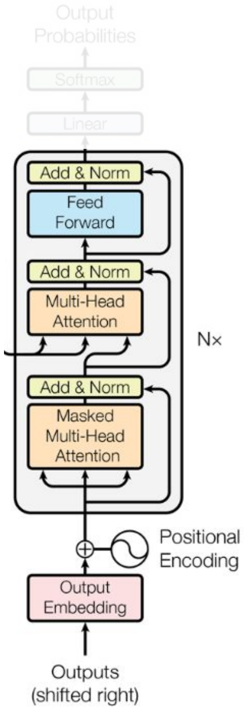
$\mathbb{R}^{m \times d_1}$

Masked Multi-Head Attention

$\mathbb{R}^{m \times d_1}$

Embedded target sequence

$\mathbb{R}^{m \times d_1}$



Target sequence
(<bos>, x_1, …, x_m)

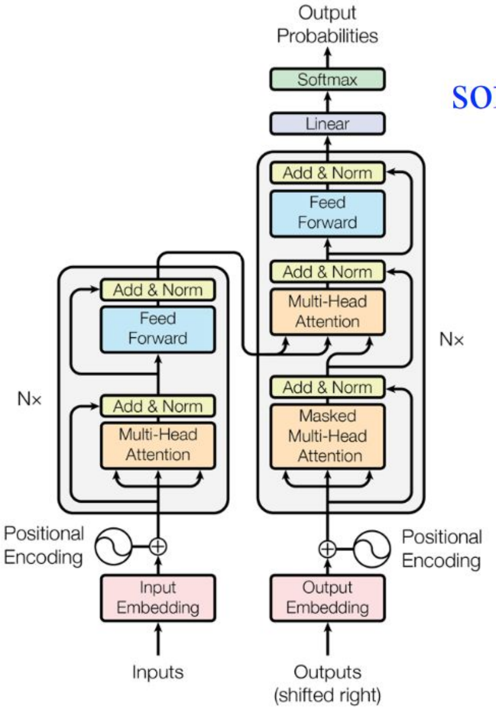
Add & Norm:

$$\mathrm{LayerNorm}(x + \mathrm{Sublayer}(x))$$

LayerNorm

$$\boxed{y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta}$$

Transformer: Final output



$$\text{softmax}(\mathbf{W}_o \mathbf{h}_i)$$

Compute transformation over **concatenated states**