Autoregressive models and attention

Instructor: Seunghoon Hong

Course logistics

- Claim session for midterm
  - Time: 1-3pm, Friday (Nov. 3rd)
  - Location: E3-1, 3428

Recap: objective of deep generative models

- Learning a model that its outputs follow the true data distribution

$$G_\theta \sim P(X)$$

Generated images from G

True Images X

Recap: objective of deep generative models

- Maximum Likelihood Estimation (MLE)

$$\hat{\theta} = \arg\max_{\theta \in \Theta} \sum_{x_i \in \mathcal{X}} \boxed{\log p_\theta(x_i)}$$ Log Likelihood

$$\leftrightarrow \arg\max_{\theta \in \Theta} \prod_{x_i \in \mathcal{X}} \boxed{p_\theta(x_i)}$$ Likelihood

Find model parameters that maximize the probability of sampling training data (likelihood)

$$\leftrightarrow \arg\min_{\theta \in \Theta} \sum_{x_i \in \mathcal{X}} -\log p_\theta(x_i)$$

In practice, we minimize the negative log likelihood for gradient descent

Recap: Challenges in evaluating likelihood

$$\hat{\theta} = \arg\min_{\theta \in \Theta} \sum_{x_i \in \mathcal{X}} -\log \boxed{p_\theta(x_i)}$$

- For high-dimensional data, it is difficult to optimize the joint distribution at once

$$x_i = [x_i^1, x_i^2, x_i^3, \ldots, x_1^d] \in \mathbb{R}^d$$
$$p(x_i) = p(x_i^1, x_i^2, x_i^3, \ldots, x_1^d)$$

- Examples of high-dimensional data
  - Image (d = number of pixels)
  - Sentence (d = length of sentence)

Recap: Auto-Regressive Model (AR)

- Factorizing the likelihood via **chain rule**

$$p(a, b) = p(a|b)p(b)$$

Recap: Auto-Regressive Model (AR)

- Factorizing the likelihood via chain rule

$$
\begin{aligned}
p_\theta(x) &= p_\theta(x_1, x_2, x_3, ..., x_T) \\
&= p_\theta(x_T | x_1, x_2, ..., x_{T-1}) \boxed{p_\theta(x_1, x_2, ..., x_{T-1})} \\
&= \prod_{t=1}^{T} p_\theta(x_t | x_1, ..., x_{t-1})
\end{aligned}
$$

apply recursively

Auto-Regressive Model (AR)

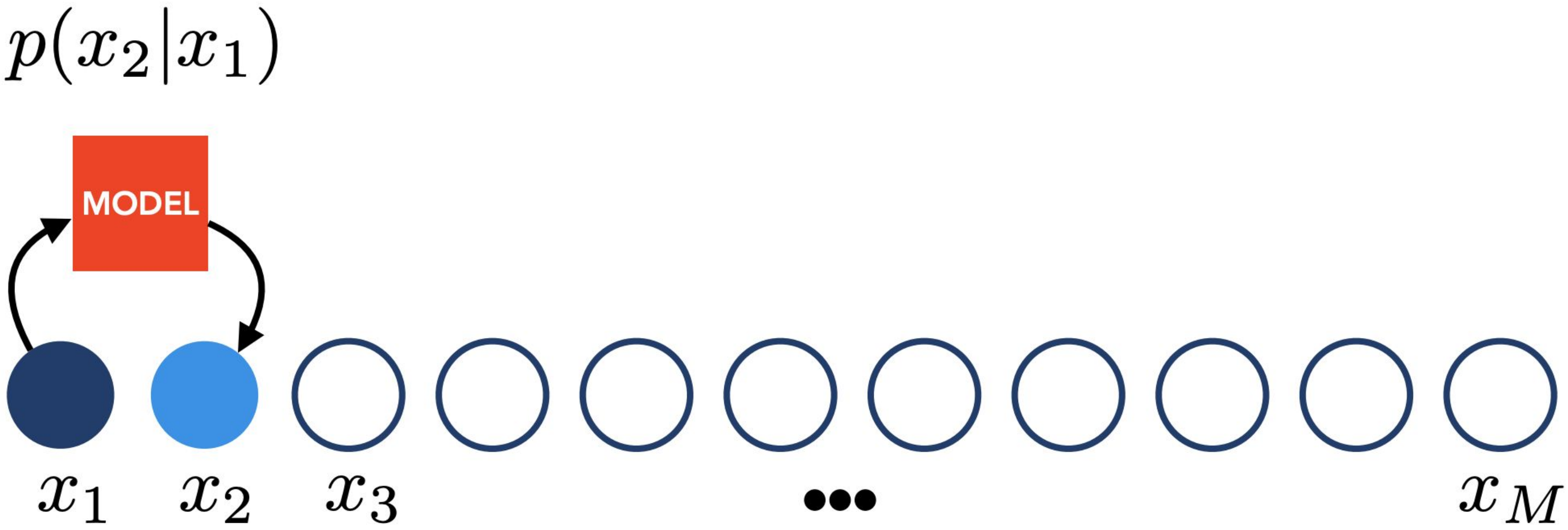
$$p_\theta(x) = \prod_{t=1}^{T} p_\theta(x_t | x_1, ..., x_{t-1})$$

$$p(x_1)$$

Slide credit: http://www.yisongyue.com/courses/cs155/2018_winter/lectures/Lecture_17.pdf

Auto-Regressive Model (AR)

$$p_\theta(x) = \prod_{t=1}^{T} p_\theta(x_t | x_1, ..., x_{t-1})$$



Slide credit: http://www.yisongyue.com/courses/cs155/2018_winter/lectures/Lecture_17.pdf

Auto-Regressive Model (AR)

$$p_\theta(x) = \prod_{t=1}^{T} p_\theta(x_t | x_1, ..., x_{t-1})$$

$$p(x_3 | x_2, x_1)$$

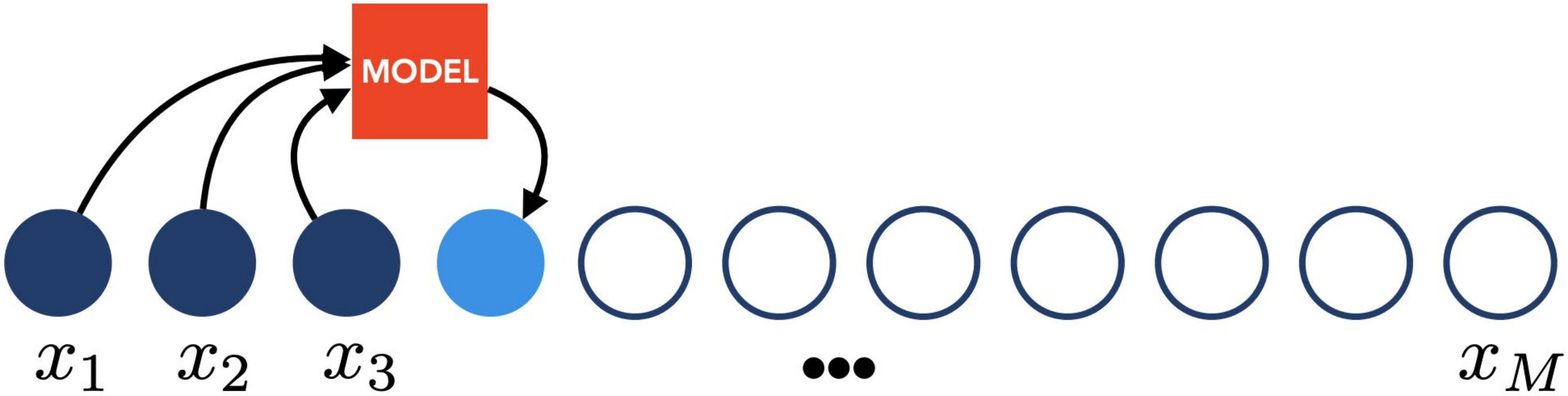
x_1 x_2 x_3 ••• x_M

Slide credit: http://www.yisongyue.com/courses/cs155/2018_winter/lectures/Lecture_17.pdf

Auto-Regressive Model (AR)

$$p_\theta(x) = \prod_{t=1}^{T} p_\theta(x_t | x_1, ..., x_{t-1})$$

$$p(x_4 | x_3, x_2, x_1)$$

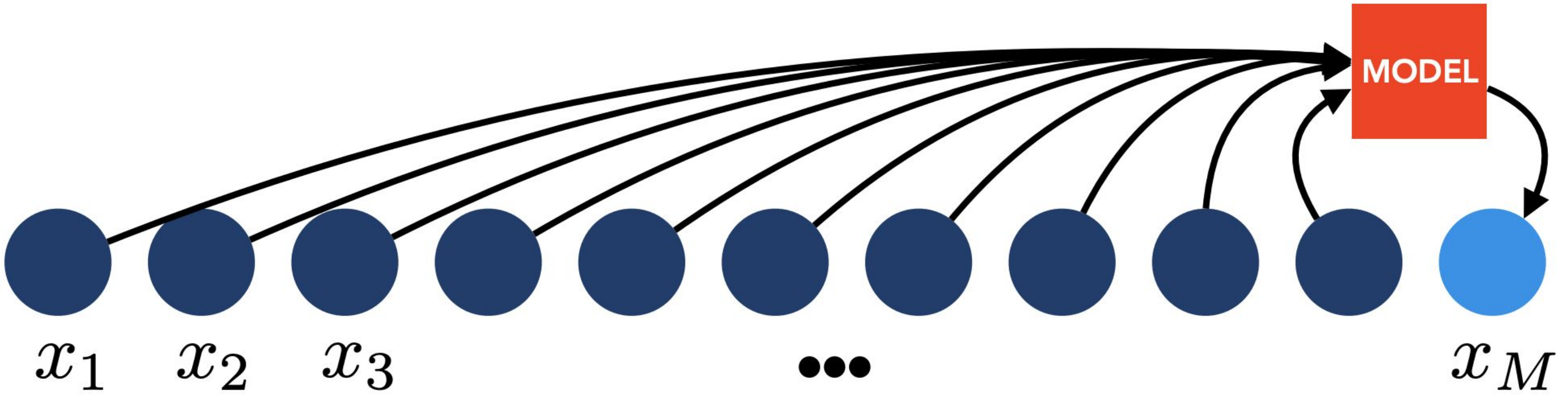


Slide credit: http://www.yisongyue.com/courses/cs155/2018_winter/lectures/Lecture_17.pdf

Auto-Regressive Model (AR)

$$p_\theta(x) = \prod_{t=1}^{T} p_\theta(x_t | x_1, ..., x_{t-1})$$

$$p(x_M | x_{M-1}, \ldots, x_1)$$



Slide credit: http://www.yisongyue.com/courses/cs155/2018_winter/lectures/Lecture_17.pdf

Recap: autoregressive model

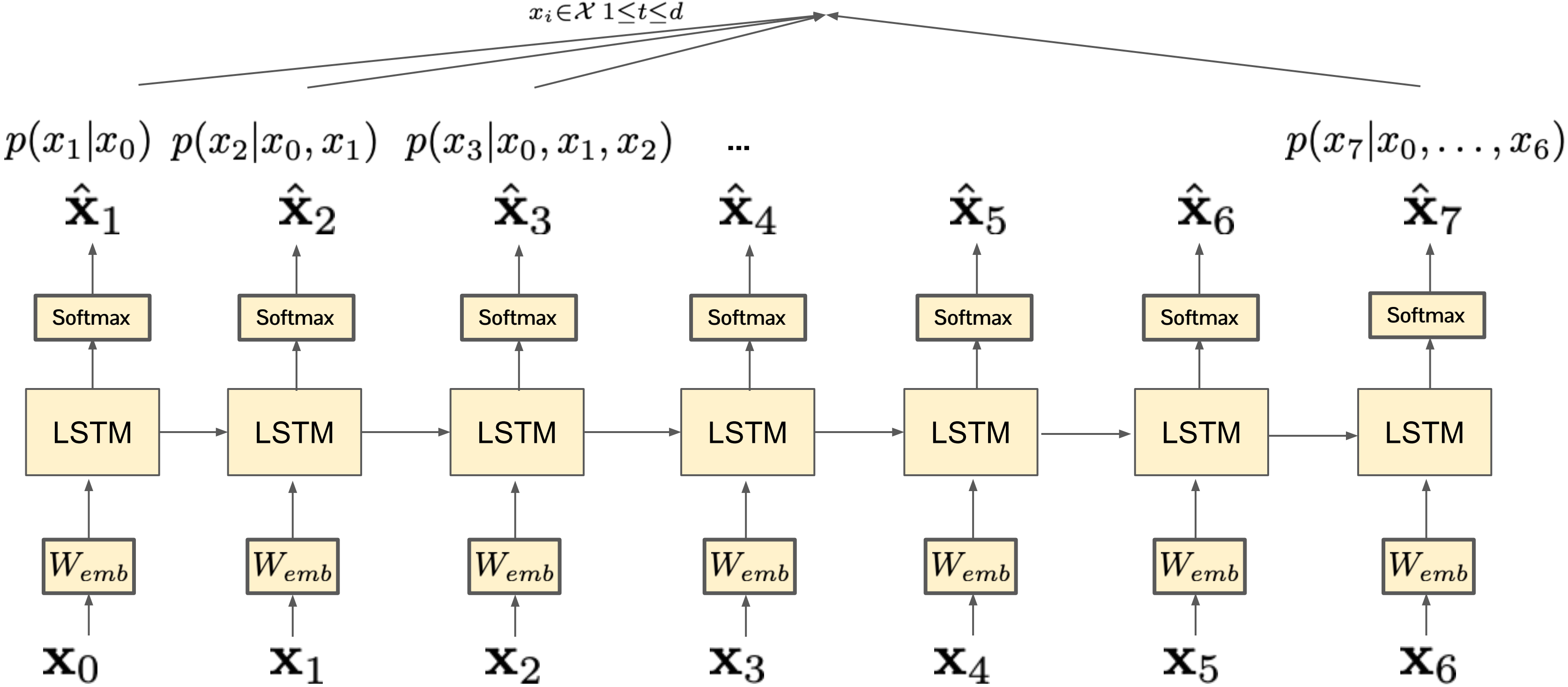
- Factorized objective function

$$\hat{\theta} = \arg\min_{\theta \in \Theta} \sum_{x_i \in \mathcal{X}} -\log p_\theta(x_i)$$

$$= \arg\min_{\theta \in \Theta} \sum_{x_i \in \mathcal{X}} \sum_{1 \leq t \leq d} -\log p_\theta(x_i^t | x_i^1, \ldots, x_i^{t-1})$$

Recap: RNN as autoregressive model

$$\sum_{x_i \in \mathcal{X}} \sum_{1 \le t \le d} - \log p_\theta(\hat{x}_i^t = x_i^t | x_i^1, \ldots, x_i^{t-1})$$

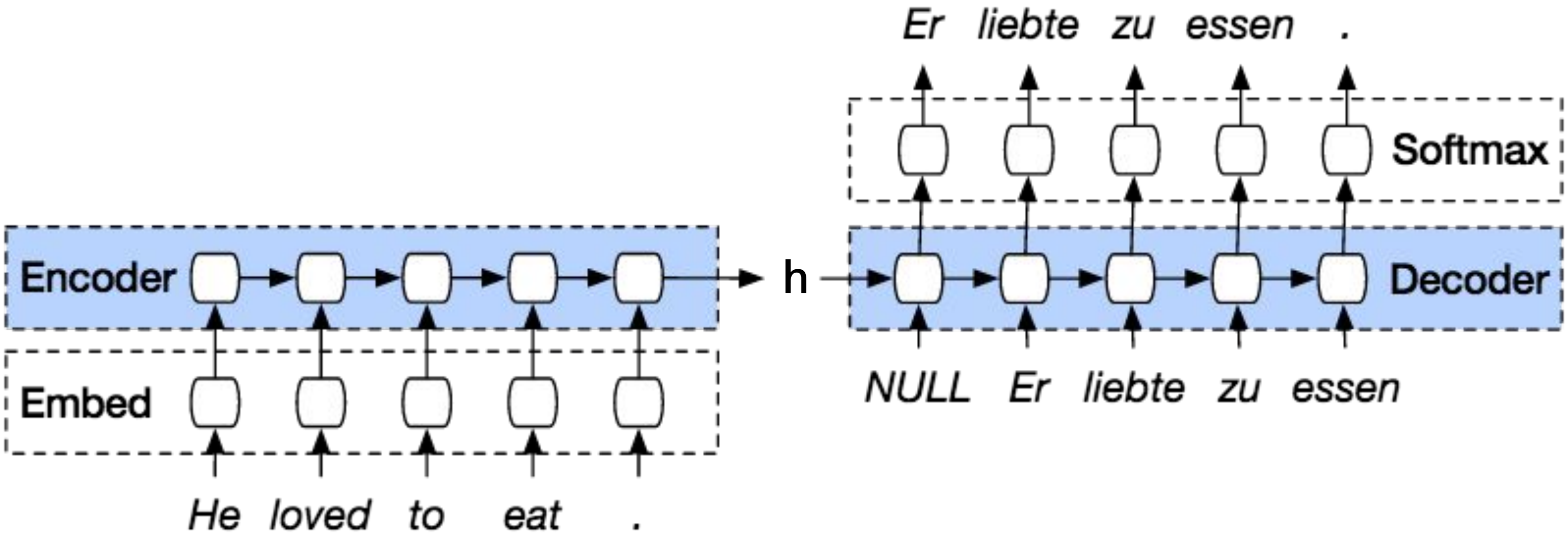


Today's agenda

- AR for machine translation
- AR with attention

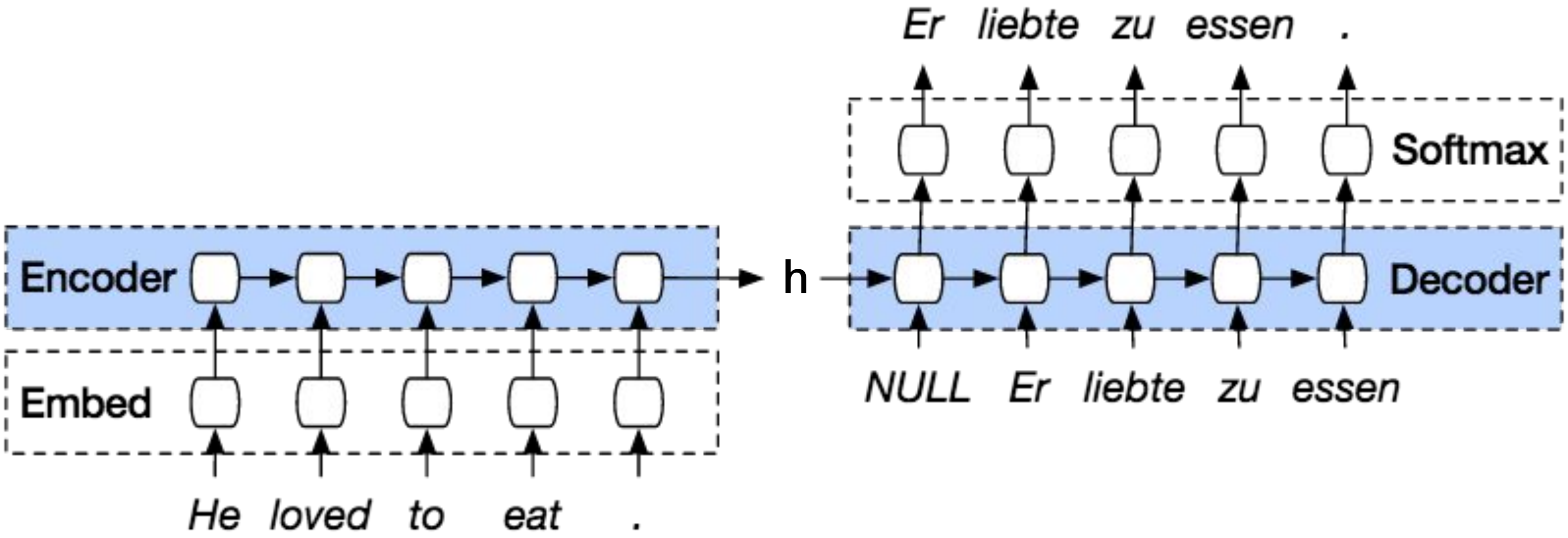
Task: machine translation

- Translating a sentence in one language to another
  - Example: English to French
- Generally, this problem is also referred to **sequence-to-sequence** generation



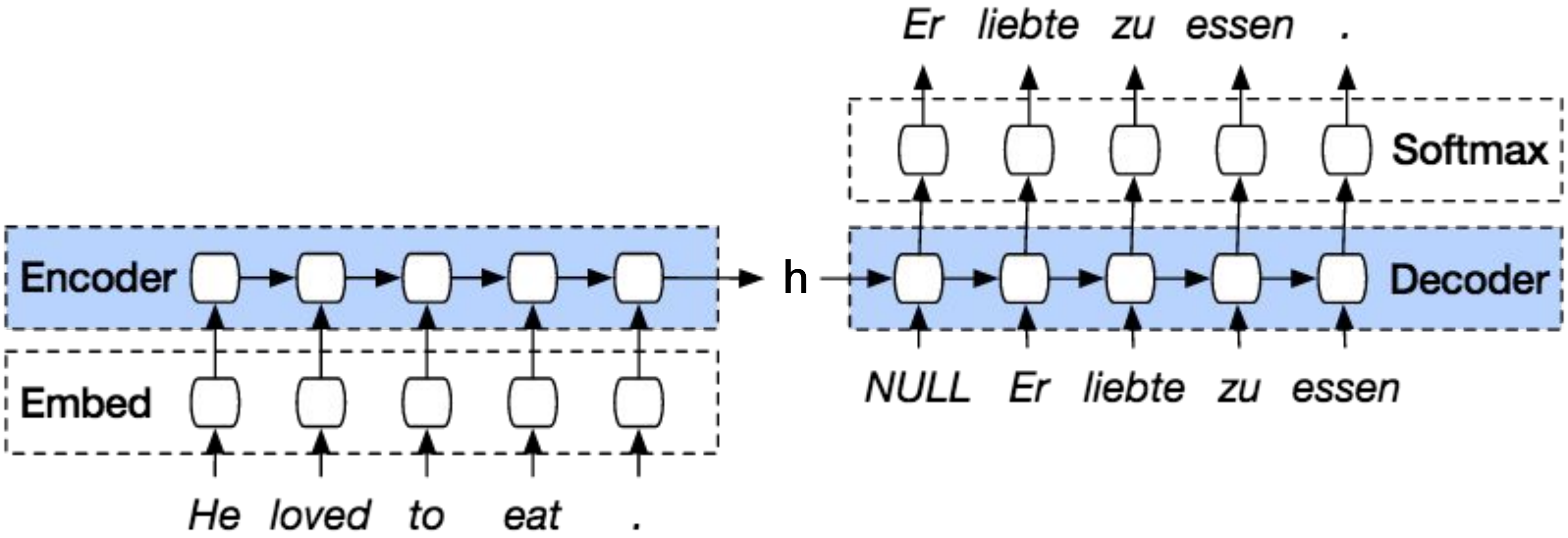
Sequence-to-sequence model

- Sequence-to-sequence model is a **conditional** autoregressive model
  - Conditioning variable: $\mathbf{x} = \{x_1, x_2, \ldots, x_M\}$
  - Decoding variable: $\mathbf{y} = \{y_1, y_2, \ldots, y_N\}$
  - Learning objective: $\theta^* = \arg\min_\theta - \sum_i \log p(\mathbf{y}_i | \mathbf{x}_{\cancel{i}})$



Sequence-to-sequence model

- A recurrent neural network with encoder-decoder architecture
  - **Encoder:** encode information of source sentence
  - **Decoder:** decode the encoded source sentence into another sequence (e.g. machine translation)



AR as Sequence-to-Sequence model

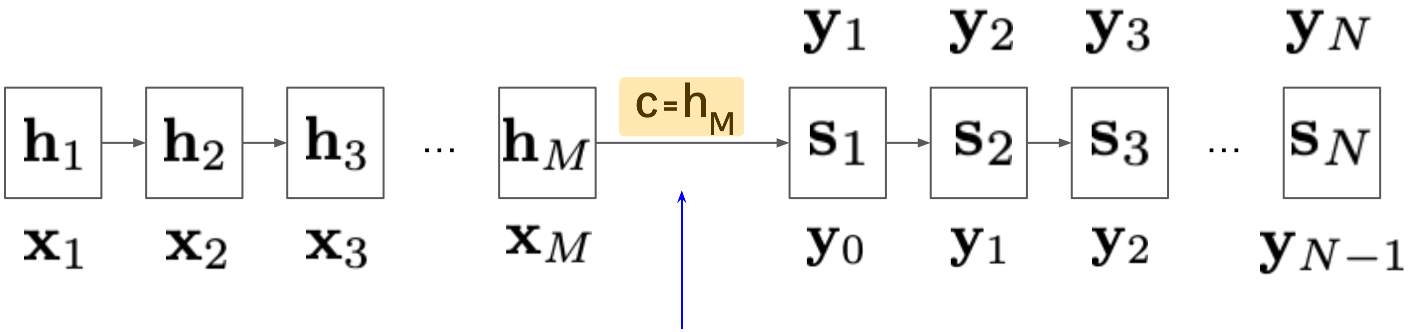
- Plain sequence-to-sequence model

Encoder

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

Decoder

$$p(y_t \mid \{y_1, \cdots, y_{t-1}\}, c) = g(y_{t-1}, s_t, c)$$

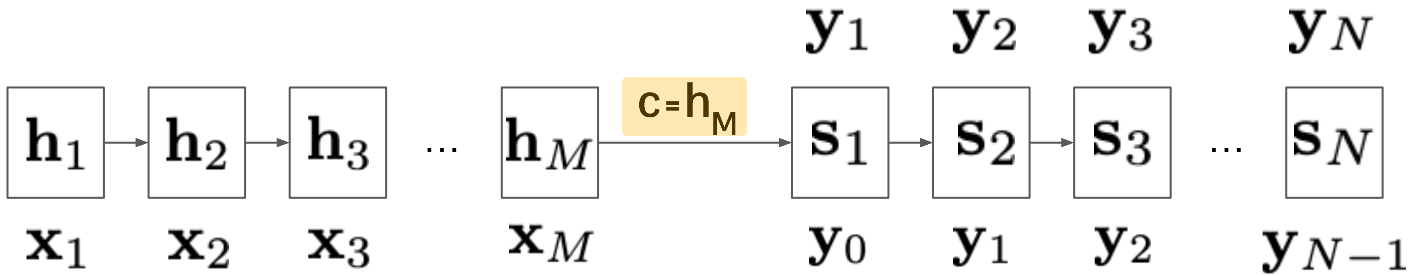


The entire source sentence is
encoded by a fixed length vector

즉, bottleneck이 되어 info. loss가 있을수도

Challenges in Sequence-to-sequence model

- Modeling long-term dependency
  - Encoder network should squash all source sentence information into a single vector **c**
  - This may lead to lossy compression, and not be appropriate to model long-term dependency



Encoder

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

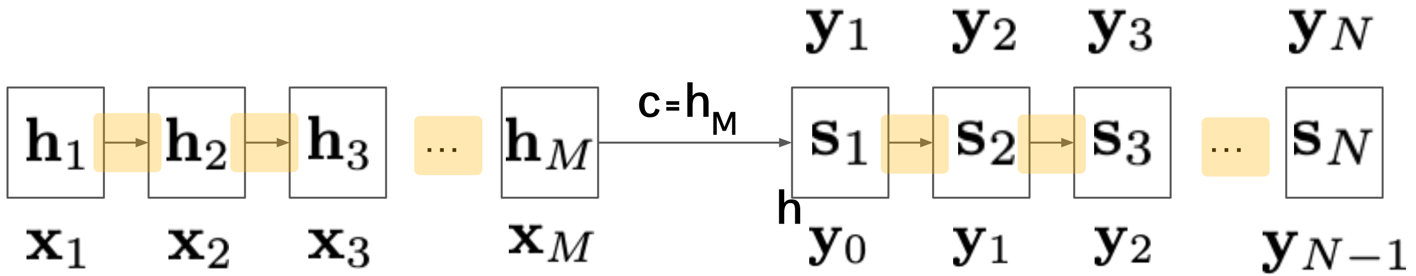
Decoder

$$p(y_t \mid \{y_1, \cdots, y_{t-1}\}, c) = g(y_{t-1}, s_t, c)$$

Challenges in Sequence-to-sequence model

- Limited to serial processing (cannot be parallelized)
  - Can be a bottleneck for efficient training

cannot parallalize



Encoder

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

Decoder

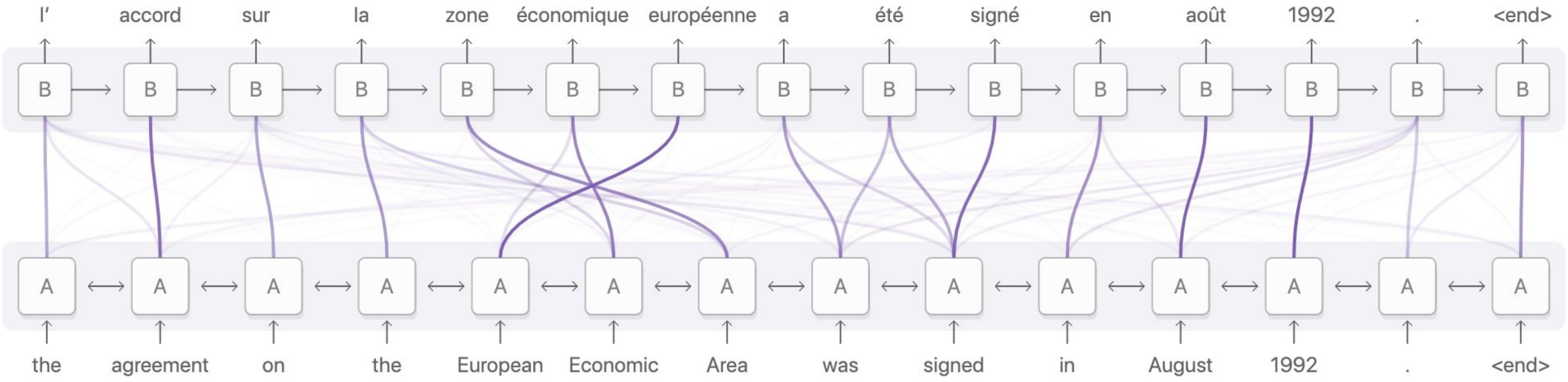
$$p(y_t \mid \{y_1, \cdots, y_{t-1}\}, c) = g(y_{t-1}, s_t, c)$$

Today’s agenda

- AR for machine translation
- **AR with attention**

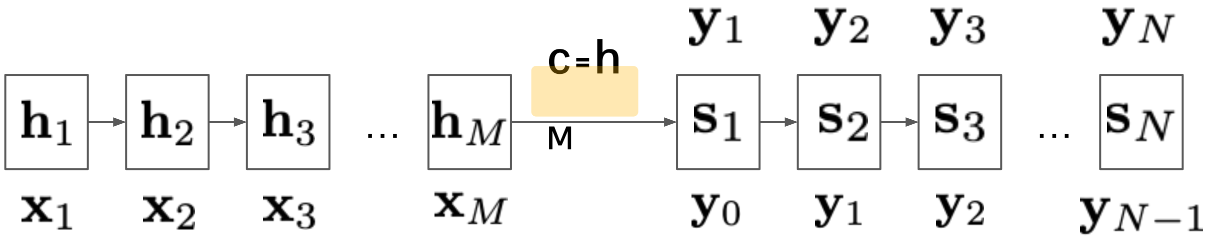
Sequence-to-sequence with attention

- Let the decoder to directly access the words in the input sentence



Sequence-to-sequence with attention

- Let the decoder to directly access the words in the input sentence



Encoder

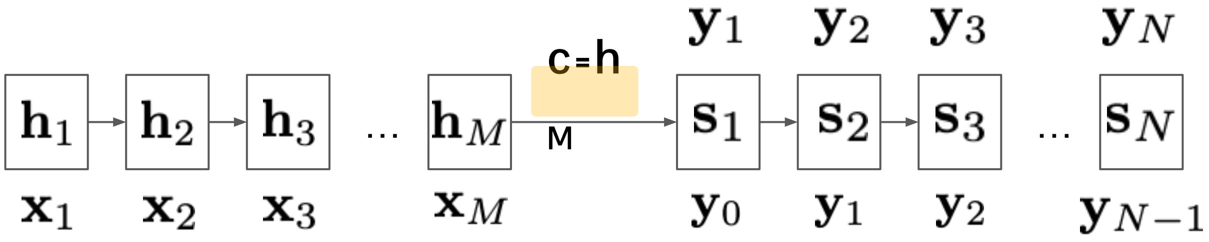
$$h_t = f\left(x_t, h_{t-1}\right)$$

Decoder

$$p(y_t \mid \{y_1, \cdots, y_{t-1}\}, c) = g(y_{t-1}, s_t, c)$$

Sequence-to-sequence with attention

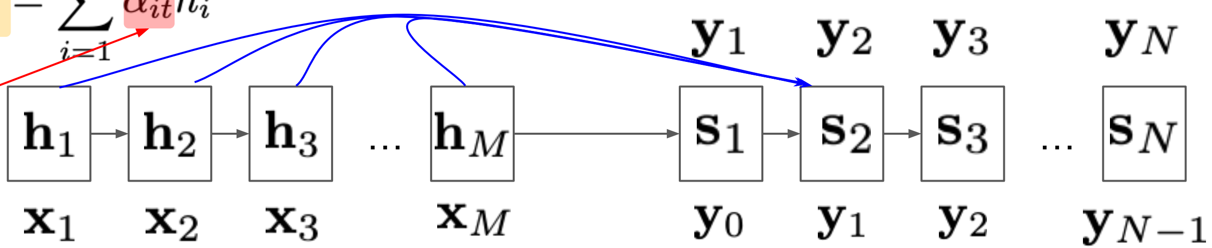
- Let the decoder to directly access the words in the input sentence



t: decoding step

$$c_t = \sum_{i=1}^{M} \alpha_{it} h_i$$

Weighted average
= attention



Encoder

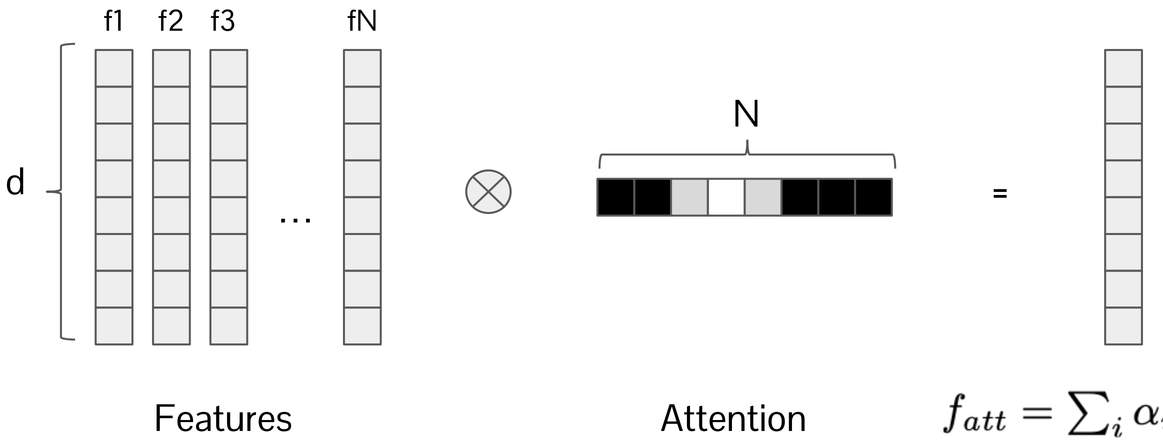
$$h_t = f\left(x_t, h_{t-1}\right)$$

Decoder

$$p(y_t | \{y_1, \dots, y_{t-1}\}, \{x_1, \dots, x_M\}) = g(y_{t-1}, s_t, c_t)$$

Recap: attention

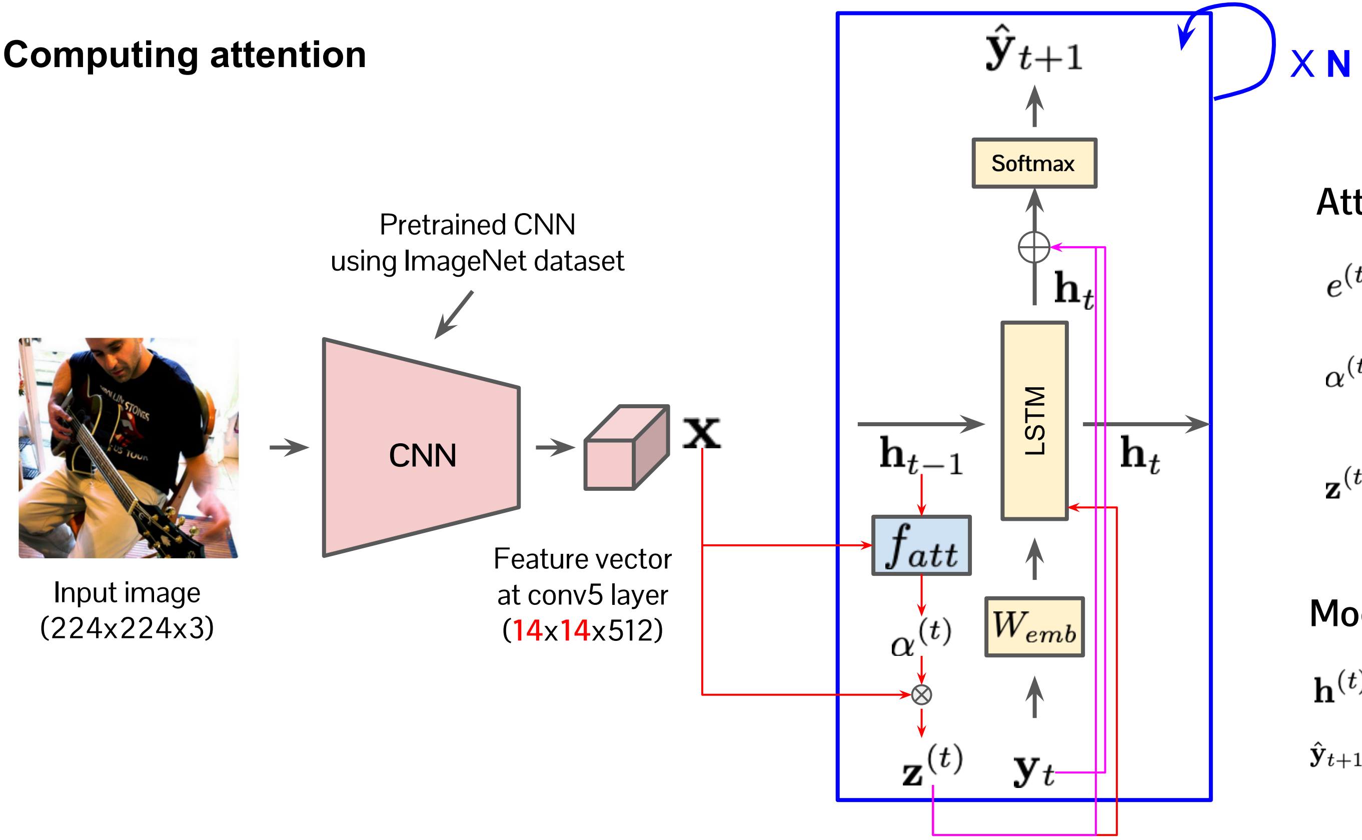
- Attention
  - A non-negative vector that summed to one $\alpha \in \mathbb{R}^N, \quad \sum_i \alpha_i = 1$
  - The size of attention vector is same as the feature that we want to apply the attention to
  - Larger value in attention means that the corresponding feature is more important than the others



$$f_{att} = \sum_i \alpha_i \mathbf{f}_i \in \mathbb{R}^d$$

Recap: attention in image captioning

Computing attention



Attention module

$$e^{(t)} = f_{att}(\mathbf{x}, \mathbf{h}_{t-1})$$

$$\alpha^{(t)} = \frac{\exp(e^{(t)})}{\sum_{i,j} \exp(e^{(t)}_{i,j})}$$

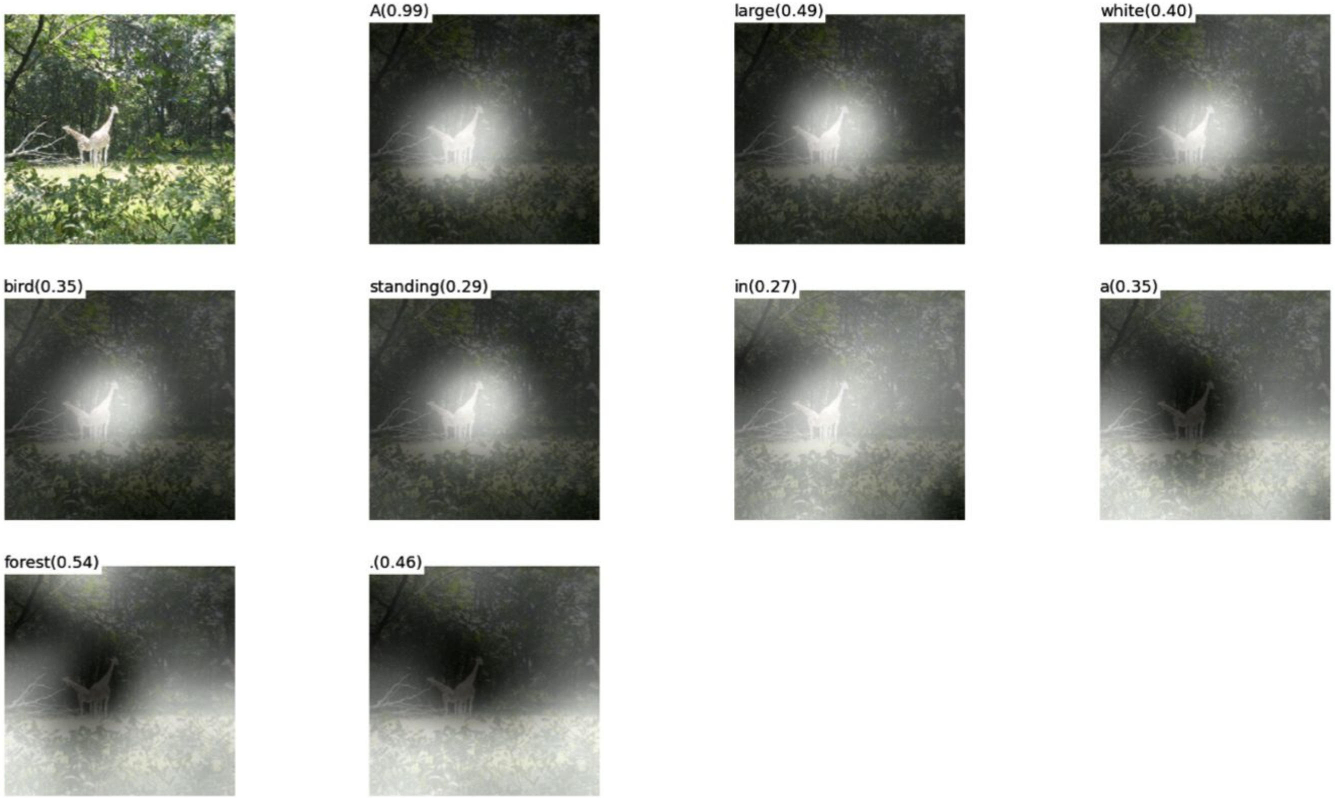
$$\mathbf{z}^{(t)} = \sum_{i,j} \alpha^{(t)}_{i,j} \mathbf{x}_{i,j}$$

Modified LSTM

$$\mathbf{h}^{(t)} = LSTM(\mathbf{h}_{t-1}, W_{emb}\mathbf{y}_t, \mathbf{z}^{(t)})$$

$$\hat{\mathbf{y}}_{t+1} = \exp(W^o(W_{emb}\mathbf{y}_t + W^h\mathbf{h}_t + W^z\mathbf{z}_t))$$

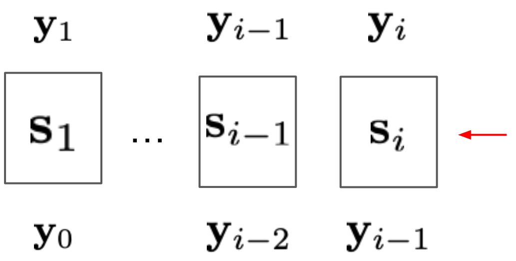
Recap: attention in image captioning



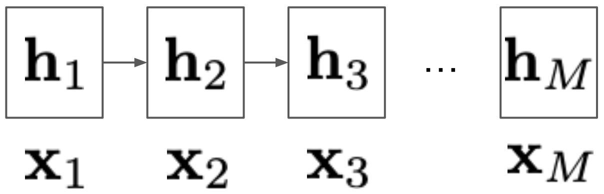
AR with attention

Decoder

$$p(y_i|y_1,\ldots,y_{i-1},\mathbf{x}) = g(y_{i-1},s_i,c_i)$$



We will compute the attention at ith step



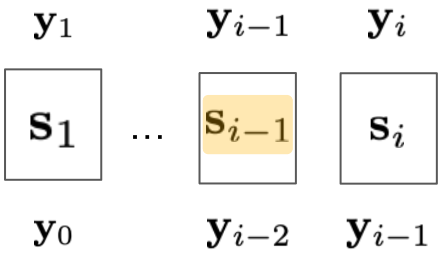
Encoder

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

AR with attention

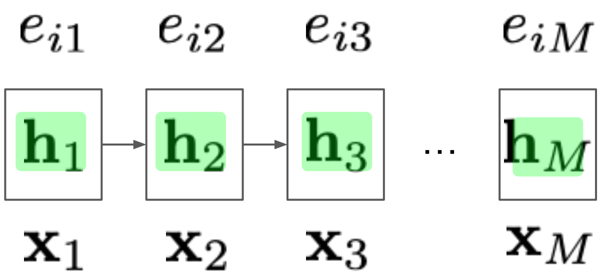
Decoder

$$p(y_i|y_1,\ldots,y_{i-1},\mathbf{x}) = g(y_{i-1}, s_i, c_i)$$



Attention

$$e_{ij} = a(s_{i-1}, h_j)$$



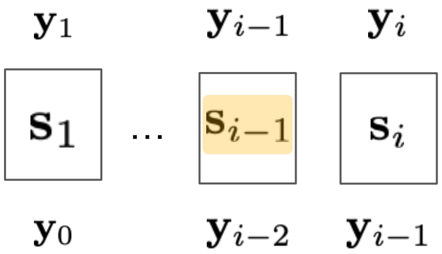
Encoder

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

AR with attention

Decoder

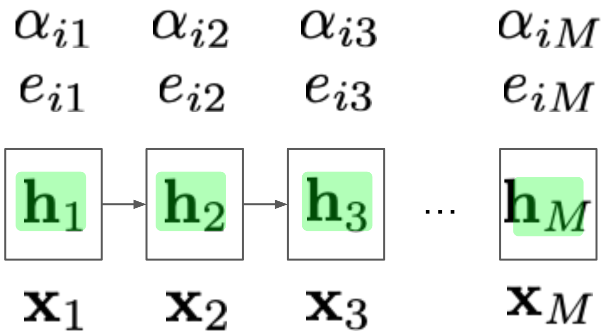
$$p(y_i|y_1,\ldots,y_{i-1},\mathbf{x}) = g(y_{i-1},s_i,c_i)$$



Attention

$$e_{ij} = a(s_{i-1}, h_j)$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{M}\exp(e_{ik})}$$



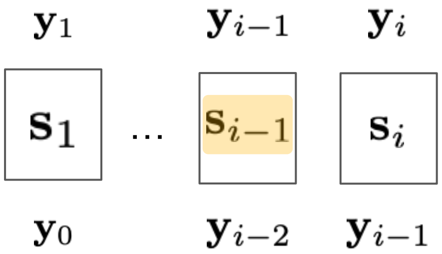
Encoder

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

AR with attention

Decoder

$$p(y_i|y_1, \ldots, y_{i-1}, \mathbf{x}) = g(y_{i-1}, s_i, c_i)$$

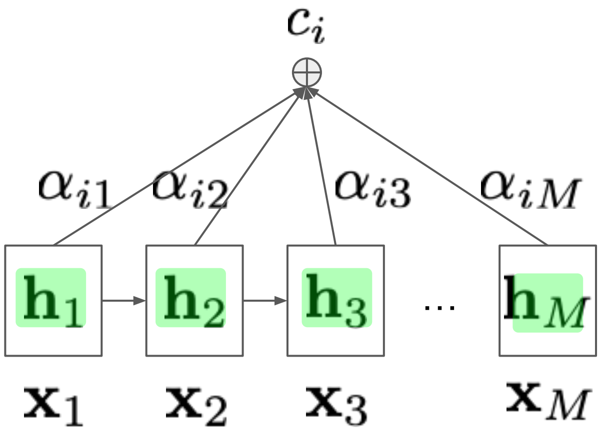


Attention

$$e_{ij} = a(s_{i-1}, h_j)$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{M} \exp(e_{ik})}$$

$$c_i = \sum_{j=1}^{M} \alpha_{ij} h_j$$



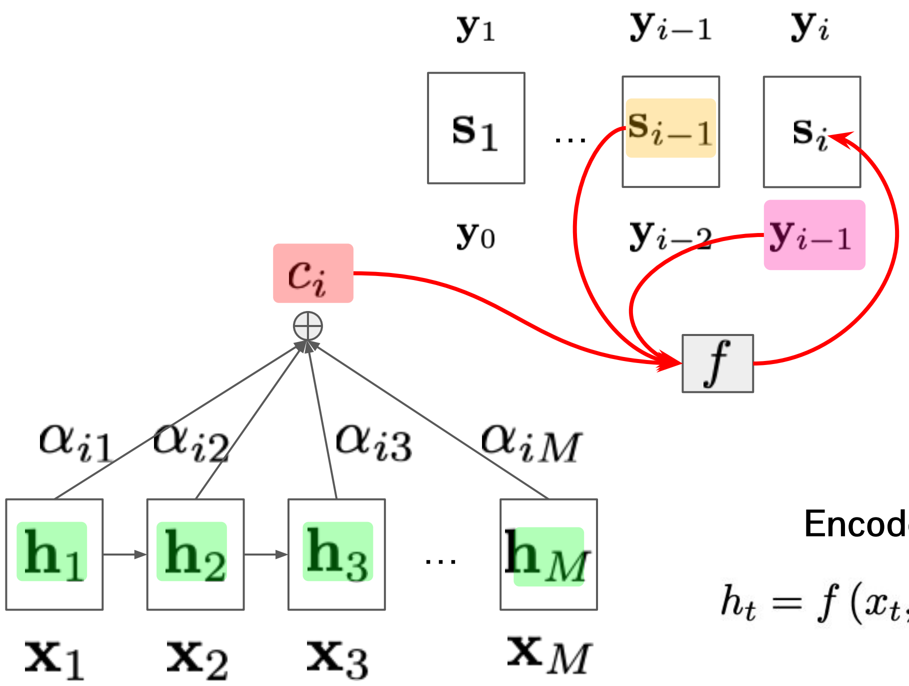
Encoder

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

AR with attention

Decoder

$$p(y_i|y_1, \ldots, y_{i-1}, \mathbf{x}) = g(y_{i-1}, s_i, c_i)$$



Encoder

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

Attention

$$e_{ij} = a(s_{i-1}, h_j)$$

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{M} \exp(e_{ik})}$$

$$c_i = \sum_{j=1}^{M} \alpha_{ij} h_j$$

$$s_i = f(s_{i-1}, y_{i-1}, c_i)$$

AR with attention

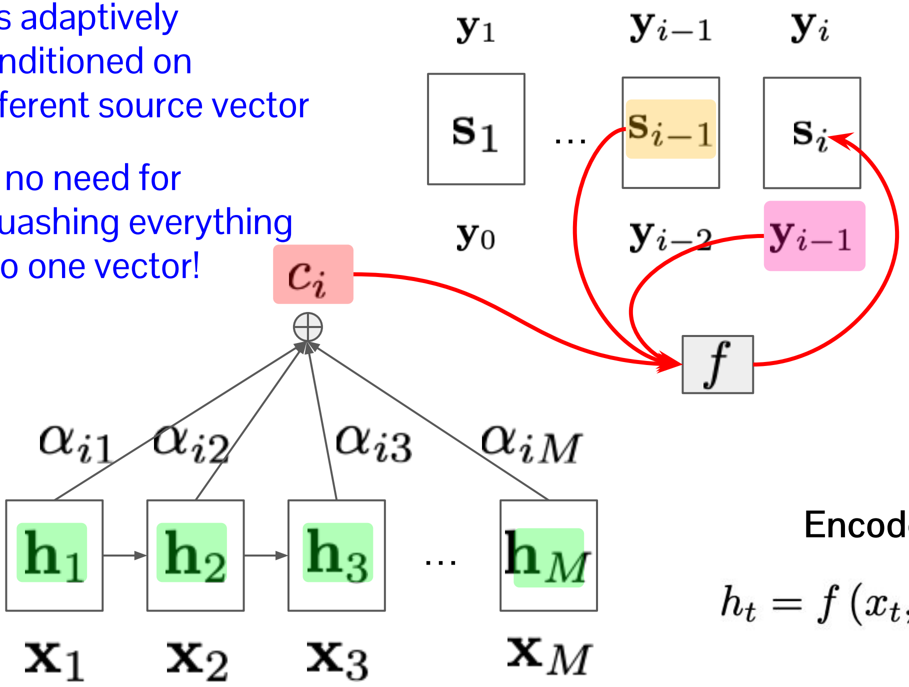
decoding step인 i가 뭐든
input과의 distance가 똑같다
-> long term dependency easy

Each step in decoder, it is adaptively conditioned on different source vector

→ no need for squashing everything into one vector!

Decoder

$$p(y_i | y_1, \ldots, y_{i-1}, \mathbf{x}) = g(y_{i-1}, s_i, c_i)$$



Encoder

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

Attention

$$e_{ij} = a(s_{i-1}, h_j)$$

Arbitrary long sentence에 a 적용 가능

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{M} \exp(e_{ik})}$$

$$c_i = \sum_{j=1}^{M} \alpha_{ij} h_j$$

$$s_i = f(s_{i-1}, y_{i-1}, c_i)$$

Results

- Visualization of attention
  - English → French
  - The attention aligns the target and source words
  - Some target words are conditioned more than one source words
  - The attention is (roughly) sequentially aligned

NOTE: 언어 문법 구조에 따라 저렇게 대각선으로 안나올수도

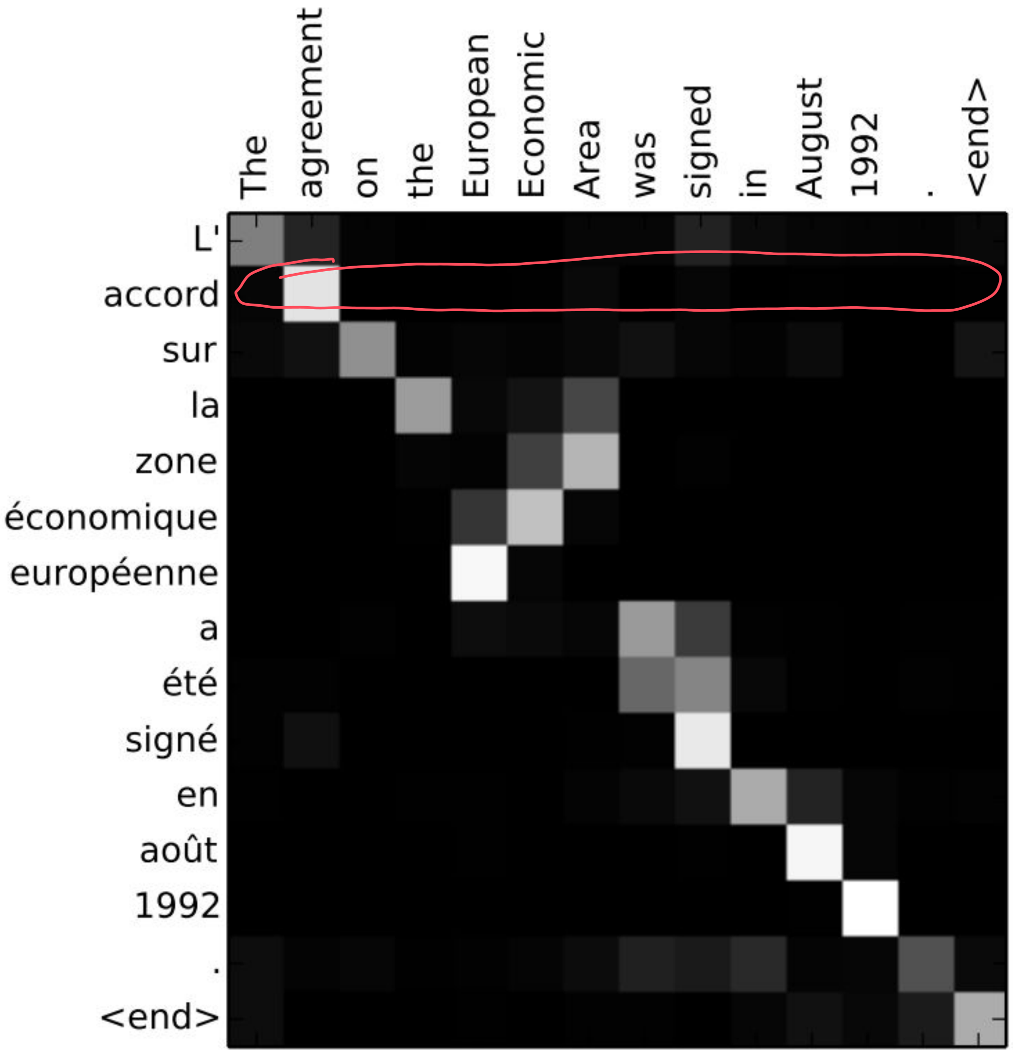
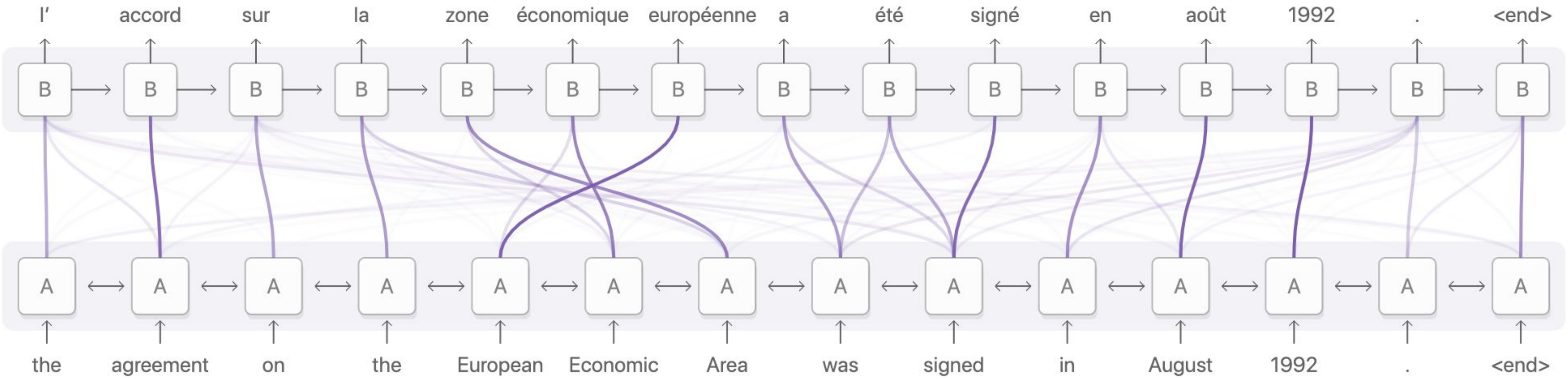


Image source: Bahdanau et al., NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE, In ICLR, 2015

Results

- Attention allows us to align the source and target sentence



Discussions: AR with attention

- Attention is turned out to be very useful in modeling long-term dependency
- Interesting observations
  - In naive AR, we used recursive connection for modeling temporal dependencies
  - Attention itself can carry the information of source during decoding (without recurrent encoder)
  - Attention is also adaptive; we can dynamically update the attention during decoding
  - Can we replace recursive connection by attention?

What is the benefit of it? → **it removes necessary for serial processing!**

Fully attention-based autoregressive model

- Transformer

Vaswani et al., Attention Is All You Need, In NIPS, 2017

Self-attention

X: input seq.

$$\text{Attn}(\mathbf{x}W_q, \mathbf{x}W_k, \mathbf{x}W_v) = \text{softmax}\left(\frac{\mathbf{x}W_q(\mathbf{x}W_k)^T}{\sqrt{d}}\right)\mathbf{x}W_v$$

Self-attention

$$\mathrm{Attn}(\mathbf{x}W_q, \mathbf{x}W_k, \mathbf{x}W_v) = \mathrm{softmax}\left(\frac{\mathbf{x}W_q(\mathbf{x}W_k)^T}{\sqrt{d}}\right)\mathbf{x}W_v$$

We have three linear projection matrices

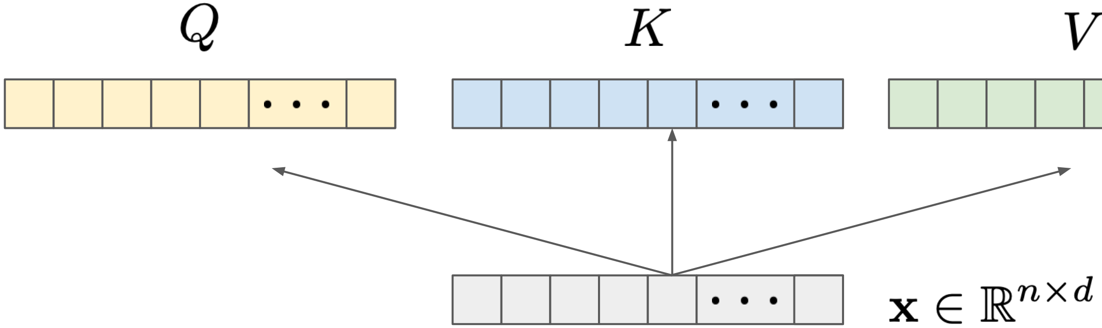
$$W_q, W_k, W_v \in \mathbb{R}^{d \times d'}$$

Self-attention

$$\mathrm{Attn}(\underbrace{\mathbf{x}W_q}_{Q}, \underbrace{\mathbf{x}W_k}_{K}, \underbrace{\mathbf{x}W_v}_{V}) = \mathrm{softmax}\left(\frac{\mathbf{x}W_q(\mathbf{x}W_k)^T}{\sqrt{d}}\right)\mathbf{x}W_v$$

We project inputs into query, key, and value

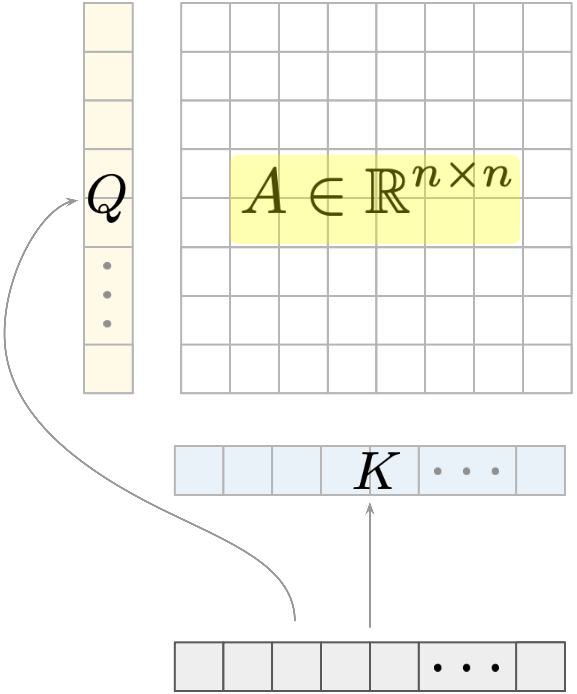
$$Q = \mathbf{x}W_q \in \mathbb{R}^{n\times d'}$$
$$K = \mathbf{x}W_k \in \mathbb{R}^{n\times d'}$$
$$V = \mathbf{x}W_v \in \mathbb{R}^{n\times d'}$$



Self-attention

Q K V

$$\mathrm{Attn}(\mathbf{x}W_q, \mathbf{x}W_k, \mathbf{x}W_v) = \mathrm{softmax}\left(\frac{\mathbf{x}W_q(\mathbf{x}W_k)^T}{\sqrt{d}}\right)\mathbf{x}W_v$$



$\mathbf{x} \in \mathbb{R}^{n \times d}$

$A = QK^T$

where

$Q \in \mathbb{R}^{n \times d'}$

$K \in \mathbb{R}^{n \times d'}$

Query and key matrices are used to compute the pairwise attention

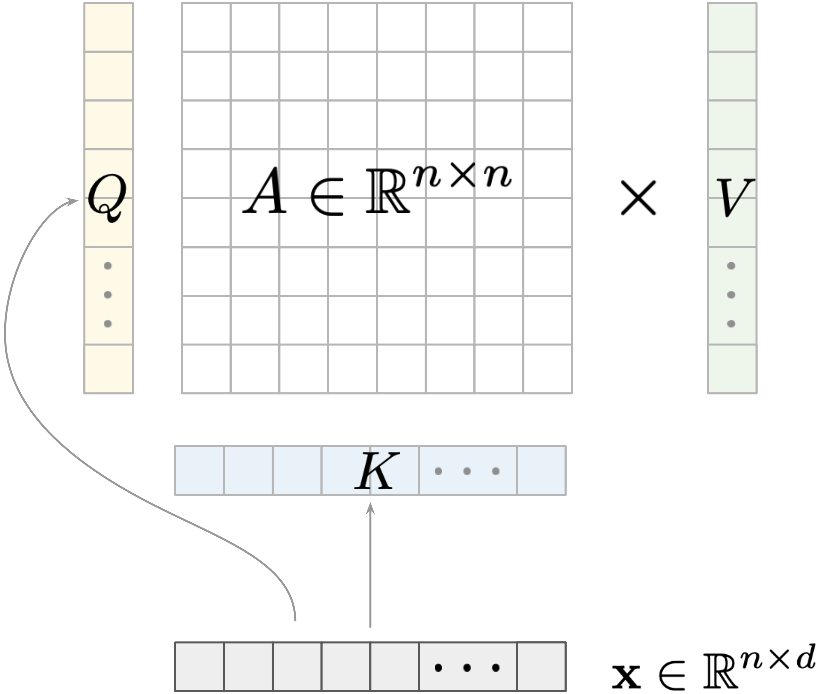
Each (i,j)th entry of attention indicates how i-th query is relevant to j-th key

The attention is row-wise sum-to-one due to softmax normalization

$\Rightarrow A_{ij} = \langle Q_i, K_j \rangle$, 즉, similarity 측정.

Self-attention

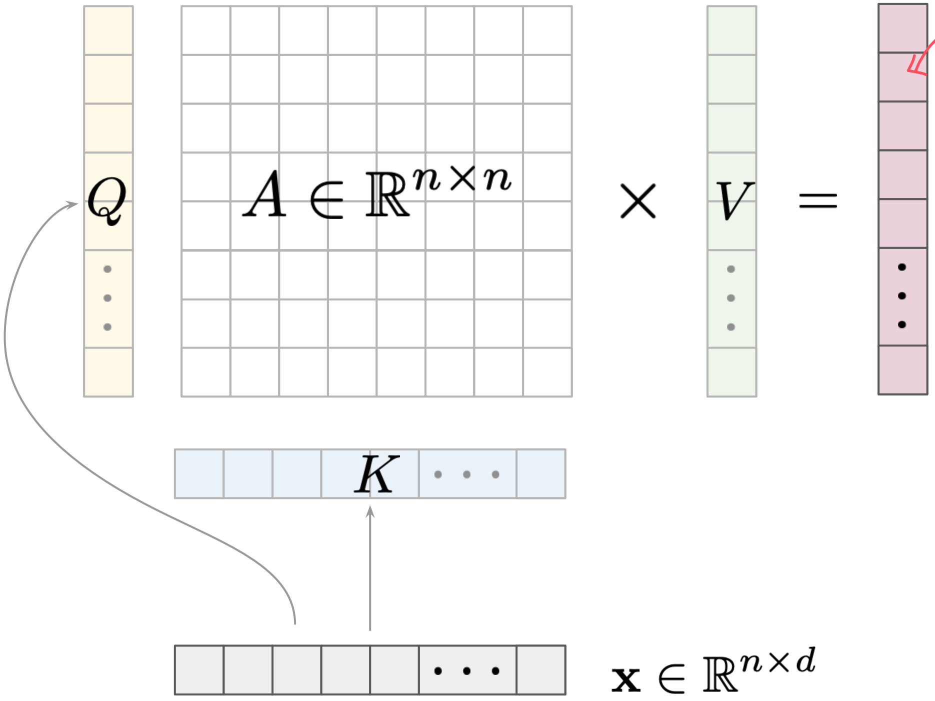
$$\mathrm{Attn}(\mathbf{x}W_q, \mathbf{x}W_k, \mathbf{x}W_v) = \mathrm{softmax}\left(\frac{\mathbf{x}W_q(\mathbf{x}W_k)^T}{\sqrt{d}}\right)\mathbf{x}W_v$$



Attention is used to take weighted average of the value matrix (matrix multiplication)

Self-attention

$$\mathrm{Attn}(\mathbf{x}W_q, \mathbf{x}W_k, \mathbf{x}W_v) = \mathrm{softmax}\left(\frac{\mathbf{x}W_q(\mathbf{x}W_k)^T}{\sqrt{d}}\right)\mathbf{x}W_v$$



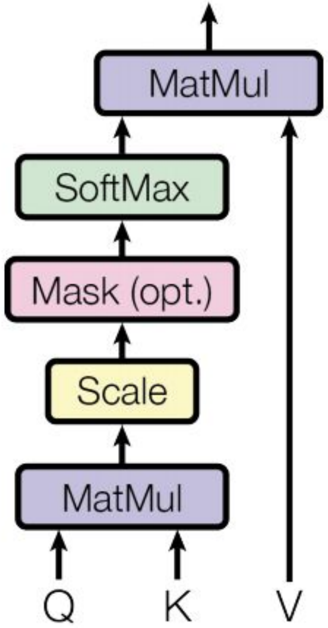
$O_i = \sum A_{i\alpha} V_\alpha$

The output is contexture encoding of input (receptive field=the whole)

$$V = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$$

Scaled Dot-Product Attention

Scaled Dot-Product Attention



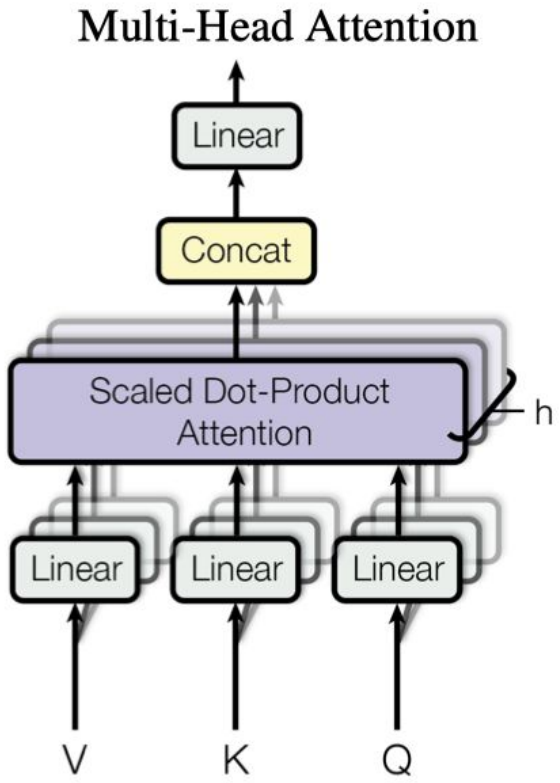
- Query, key, value attention with scaling factor

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

We normalize the logits for the softmax with the size of query and key.

It prevents the softmax function produces too sharp attention thus makes gradient stronger

Multi-head attention



- Each head corresponds to query, key, value attention (=scaled dot product attention)

$$\mathrm{MultiHead}(Q,K,V) = \mathrm{Concat}(\mathrm{head_1}, \ldots, \mathrm{head_h})W^O$$
$$\text{where } \mathrm{head_i} = \mathrm{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

- Multi-head attention computes multiple attentions using (Q_i,K_i,V_i) and aggregates them
- Note that the multi-head attention can be computed not only within the encoder and decoder features, but features between encoder and decoder as well.

Overview of Transformer architecture

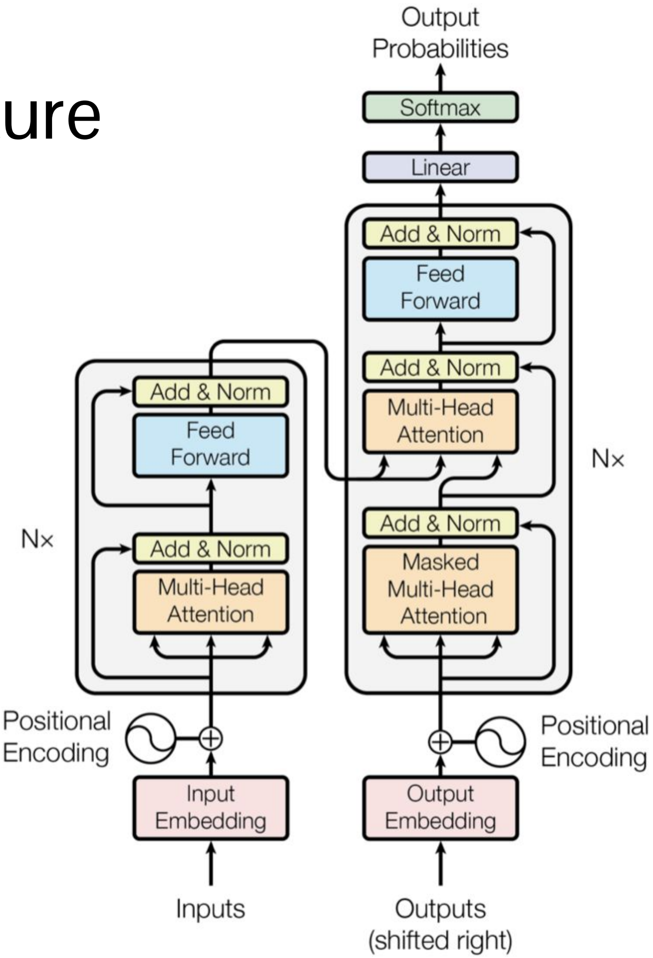


Figure 1: The Transformer - model architecture.

Transformer - Encoder

- Each word is encoded into continuous embedding

$$\mathbf{h}_i = \mathrm{FF}(\mathbf{x}_i)$$

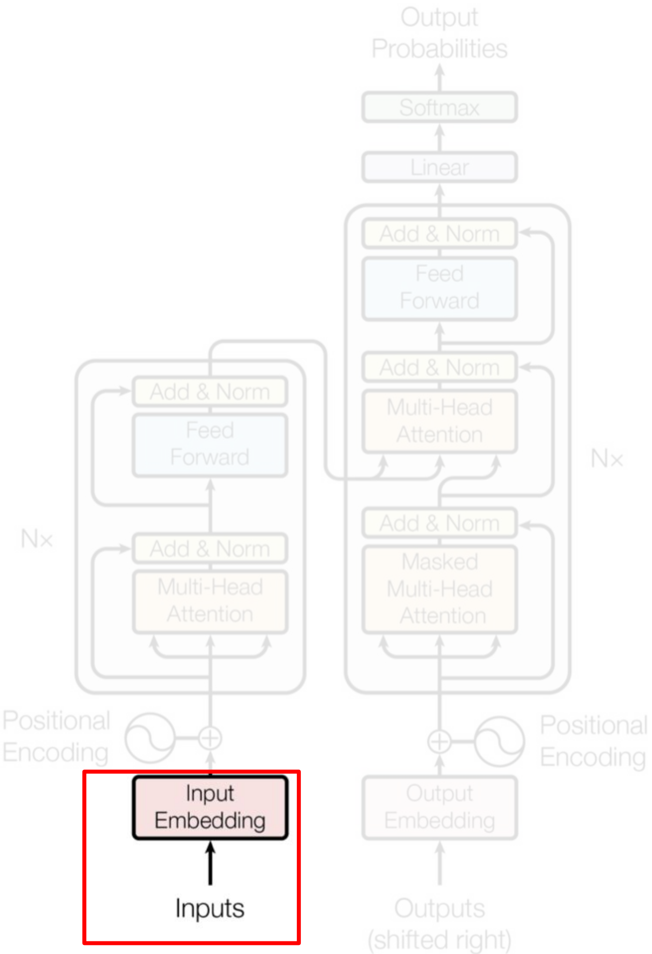


Figure 1: The Transformer - model architecture.

Transformer - Encoder

- Each word is encoded into continuous embedding
  $\mathbf{h}_i = \mathrm{FF}(\mathbf{x}_i)$
- Each word is then encoded by aggregating context within the entire source sentence
  via multi-head attention (a.k.a. **self-attention**)
  $\mathbf{h}_i = \mathrm{LayerNorm}(\mathbf{h}_i + \mathrm{Attn}(\mathbf{h}_i, \{\mathbf{h}_j\}_{\forall j \neq i}))$

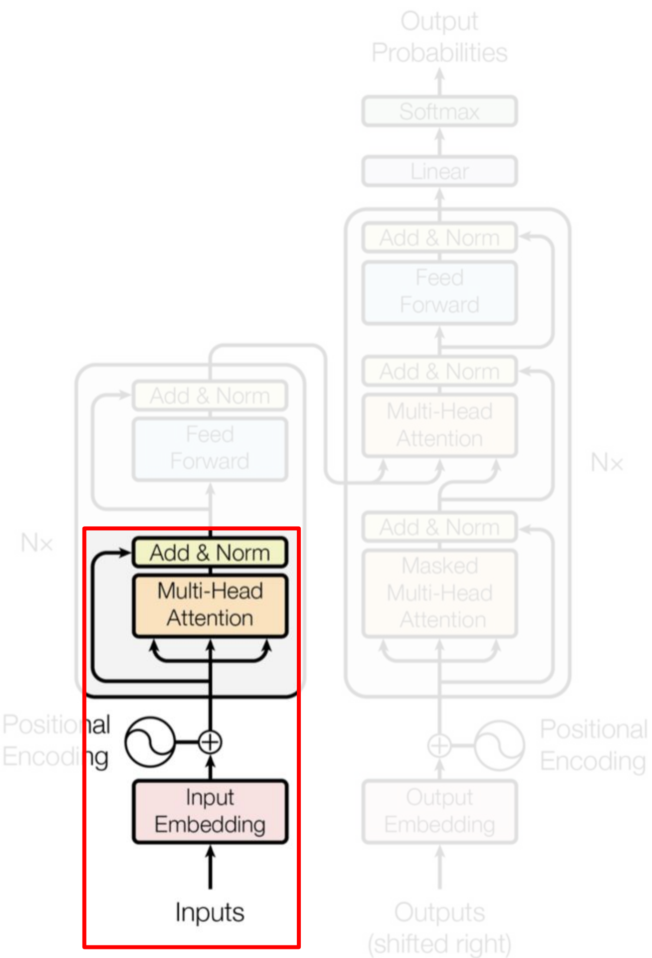


Figure 1: The Transformer - model architecture.

Transformer - Encoder

- Each word is encoded into continuous embedding
$$\mathbf{h}_i = \mathrm{FF}(\mathbf{x}_i)$$

- Each word is then encoded by aggregating context within the entire source sentence
via multi-head attention (a.k.a. **self-attention**)
$$\mathbf{h}_i = \mathrm{LayerNorm}(\mathbf{h}_i + \mathrm{Attn}(\mathbf{h}_i, \{\mathbf{h}_j\}_{\forall j \neq i}))$$

- We further encode the word with residual connection
$$\mathbf{h}_i = \mathrm{LayerNorm}(\mathbf{h}_i + \mathrm{FF}(\mathbf{h}_i))$$

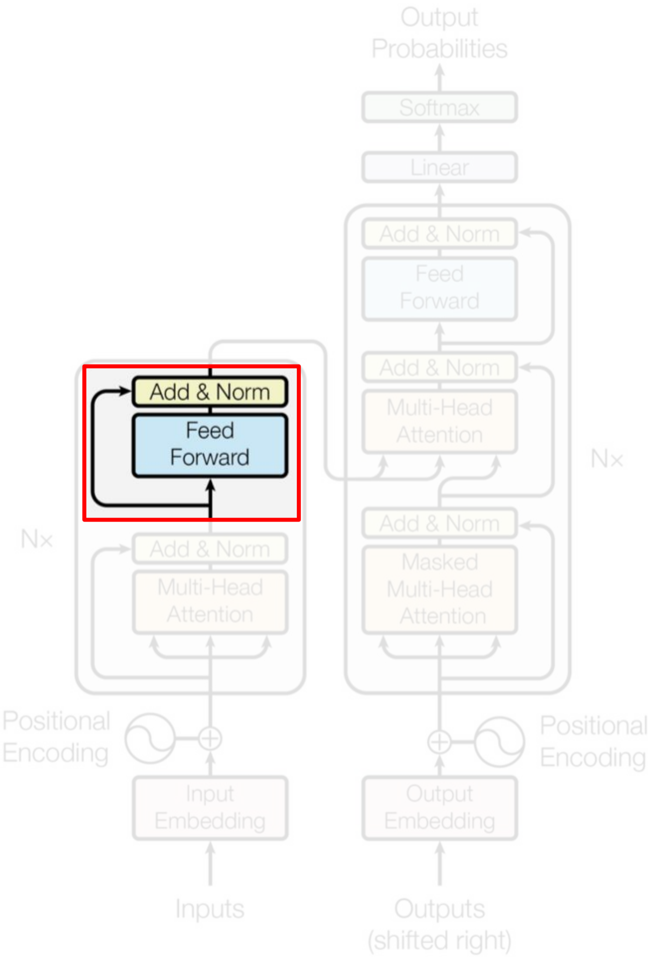


Figure 1: The Transformer - model architecture.

Transformer - Decoder

- We apply the self-attention for each target word

$$\mathbf{s}_i = \mathrm{FF}(\mathbf{y}_i)$$

$$\mathbf{s}_i = \mathrm{LayerNorm}(\mathbf{s}_i + \mathrm{Attn}(\mathbf{s}_i, \{\mathbf{s}_j\}_{\forall j<i}))$$

In decoder, we apply the attention only up to the current word (i.e., not the next words)

This is implemented via applying attention with the mask (masking the future time steps with 0)

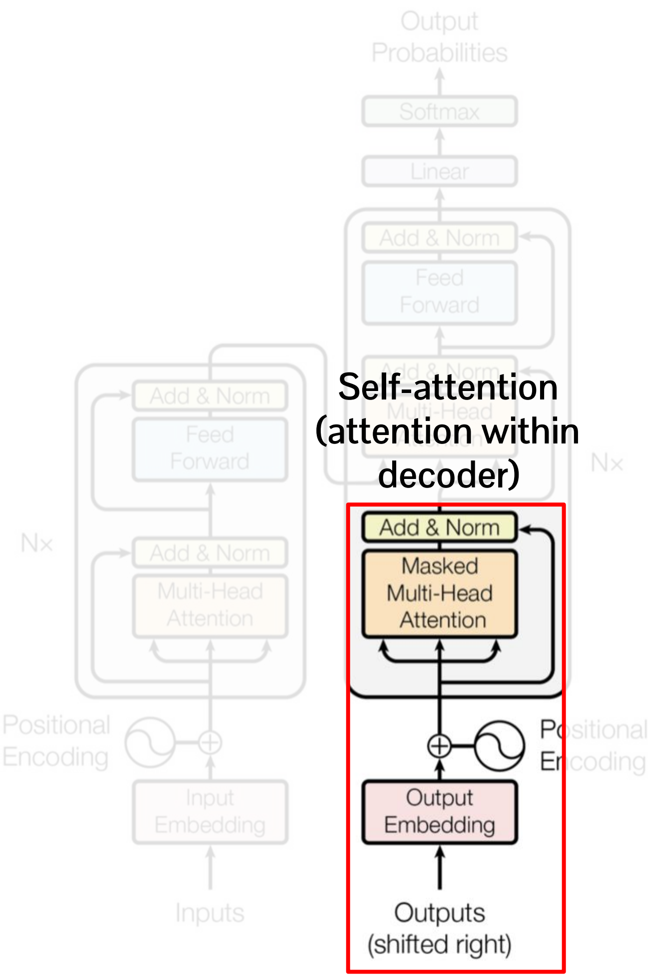


Figure 1: The Transformer - model architecture.

Transformer - Decoder

- We apply the self-attention for each target word

$$\mathbf{s}_i = \mathrm{FF}(\mathbf{y}_i)$$

$$\mathbf{s}_i = \mathrm{LayerNorm}(\mathbf{s}_i + \mathrm{Attn}(\mathbf{s}_i, \{\mathbf{s}_j\}_{\forall j<i}))$$

- The target word is encoded with source sentence via attention (a.k.a. **cross-attention**)

$$\mathbf{s}_i = \mathrm{LayerNorm}(\mathbf{s}_i + \mathrm{Attn}(\mathbf{s}_i, \{\mathbf{h}_j\}_{\forall j}))$$

All encoded words in the **source** sentence

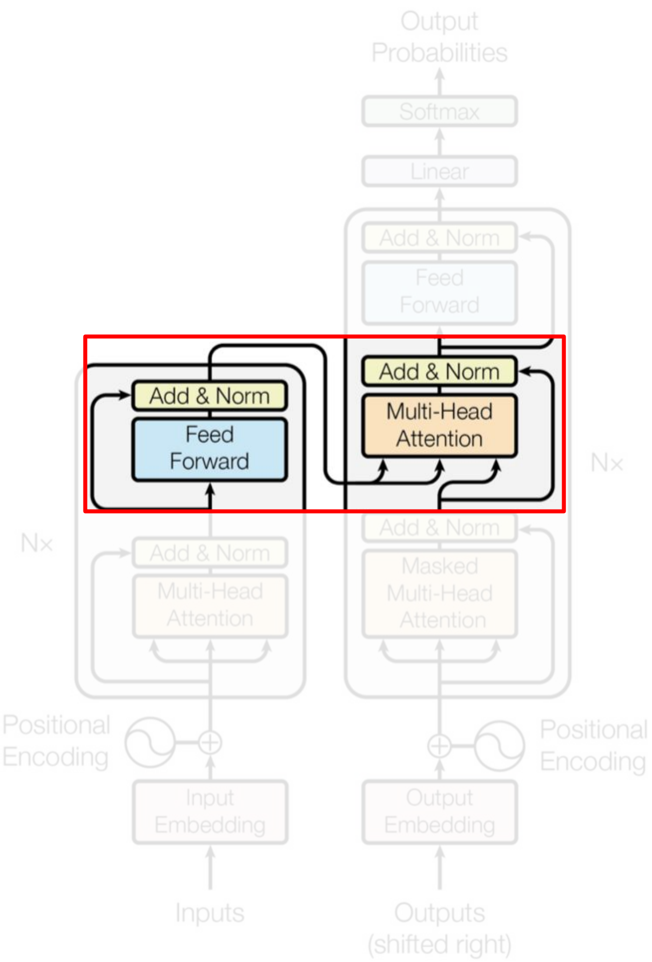


Figure 1: The Transformer - model architecture.

Transformer - Decoder

- We apply the self-attention for each target word

$$\mathbf{s}_i = \mathrm{FF}(\mathbf{y}_i)$$

$$\mathbf{s}_i = \mathrm{LayerNorm}(\mathbf{s}_i + \mathrm{Attn}(\mathbf{s}_i, \{\mathbf{s}_j\}_{\forall j<i}))$$

- The target word is encoded with source sentence via attention (a.k.a. **cross-attention**)

$$\mathbf{s}_i = \mathrm{LayerNorm}(\mathbf{s}_i + \mathrm{Attn}(\mathbf{s}_i, \{\mathbf{h}_j\}_{\forall j}))$$

- We further encode the word with residual connection

$$\mathbf{s}_i = \mathrm{LayerNorm}(\mathbf{s}_i + \mathrm{FF}(\mathbf{s}_i))$$

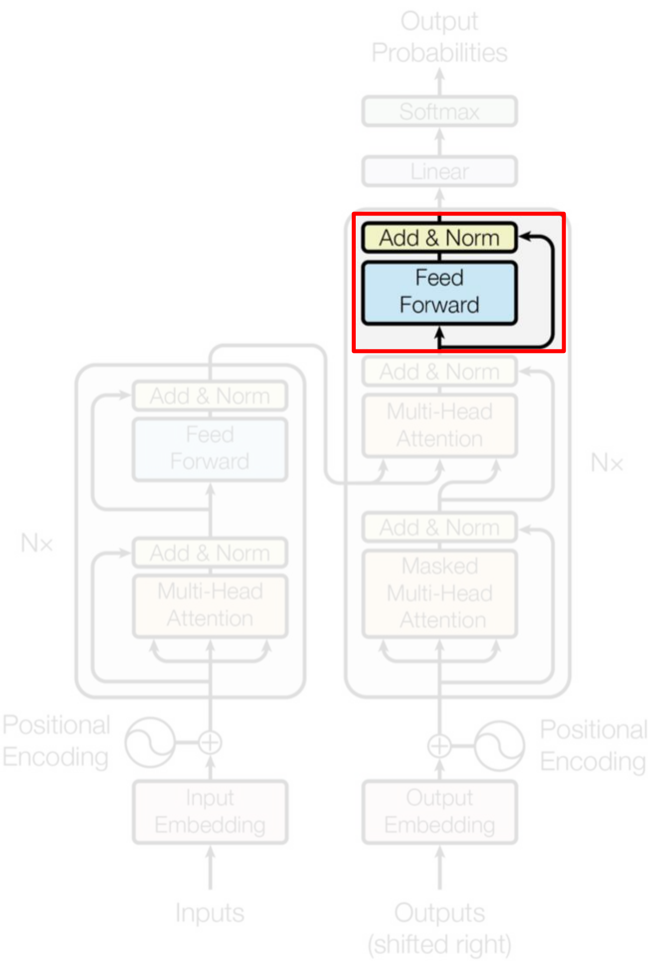


Figure 1: The Transformer - model architecture.

Transformer - Encoder & Decoder

- We stack this encoding layers multiple time for both encoder and decoder

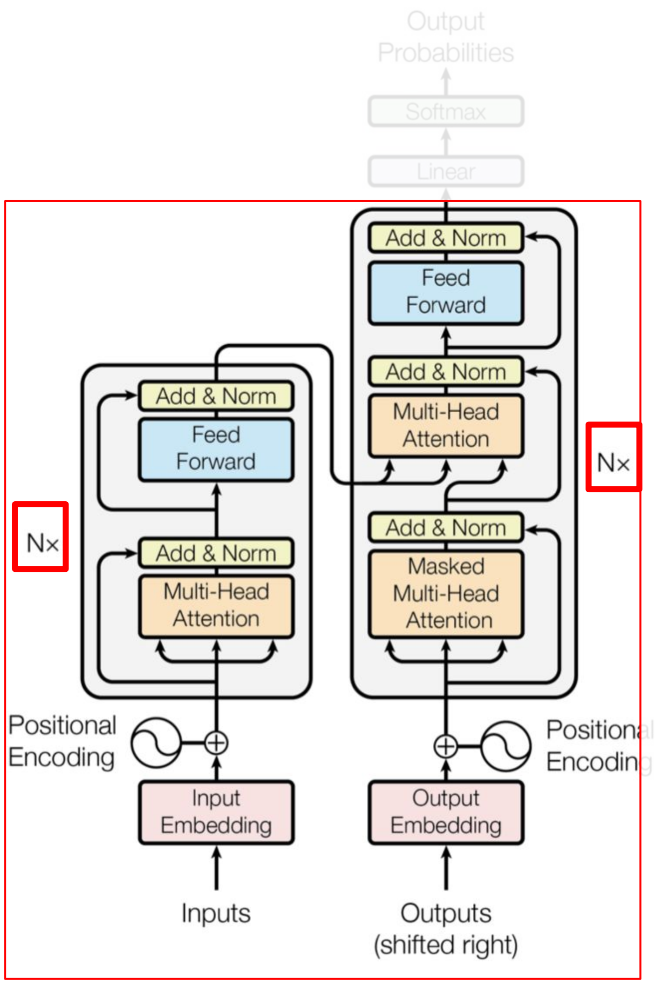


Figure 1: The Transformer - model architecture.

Transformer - Output

- The translated word is produced at each step of the decoder

$$\hat{\mathbf{y}}_{i+1} = \mathrm{Softmax}(\mathrm{FF}(\mathbf{s}_i))$$

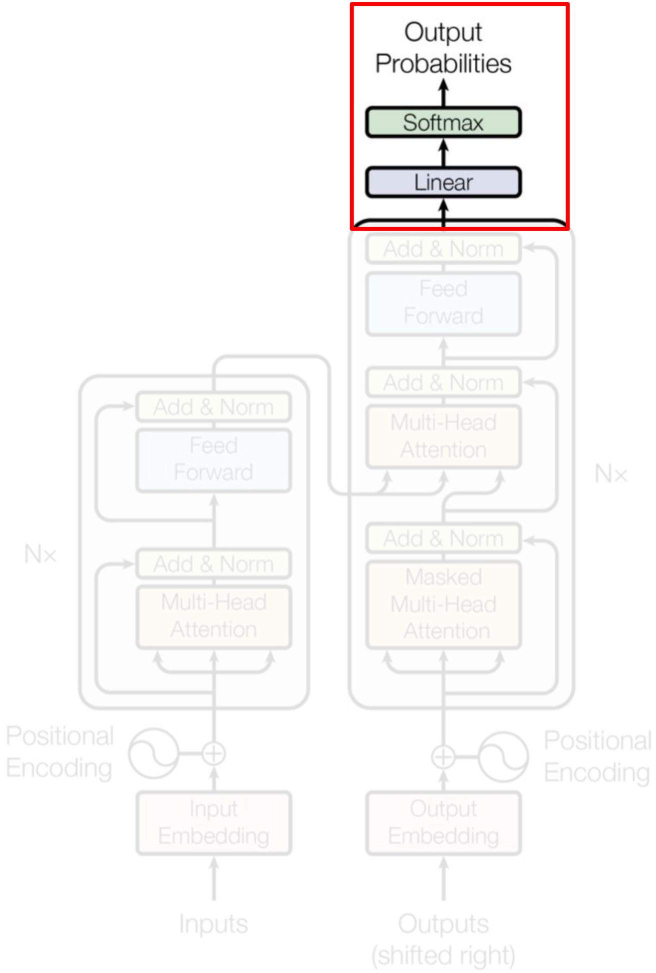


Figure 1: The Transformer - model architecture.

Transformer - Positional encoding

- When encoding both source and target words, we add extra embedding that encodes **position** of the word.

- We can use any periodic function for positional embedding, or even learn how to encode position

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

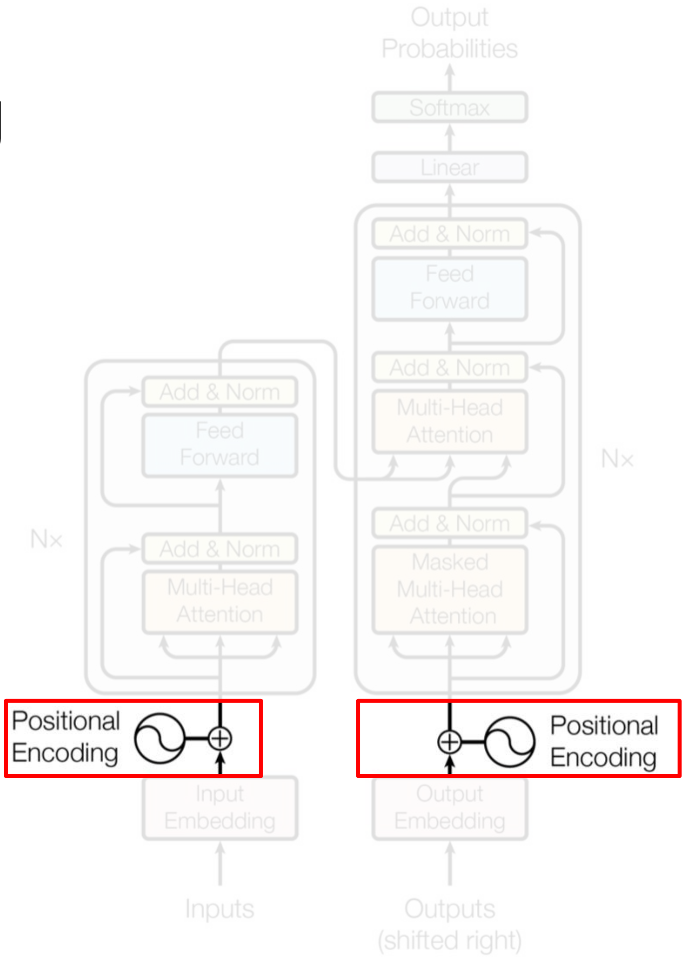


Figure 1: The Transformer - model architecture.

Overview of Transformer architecture

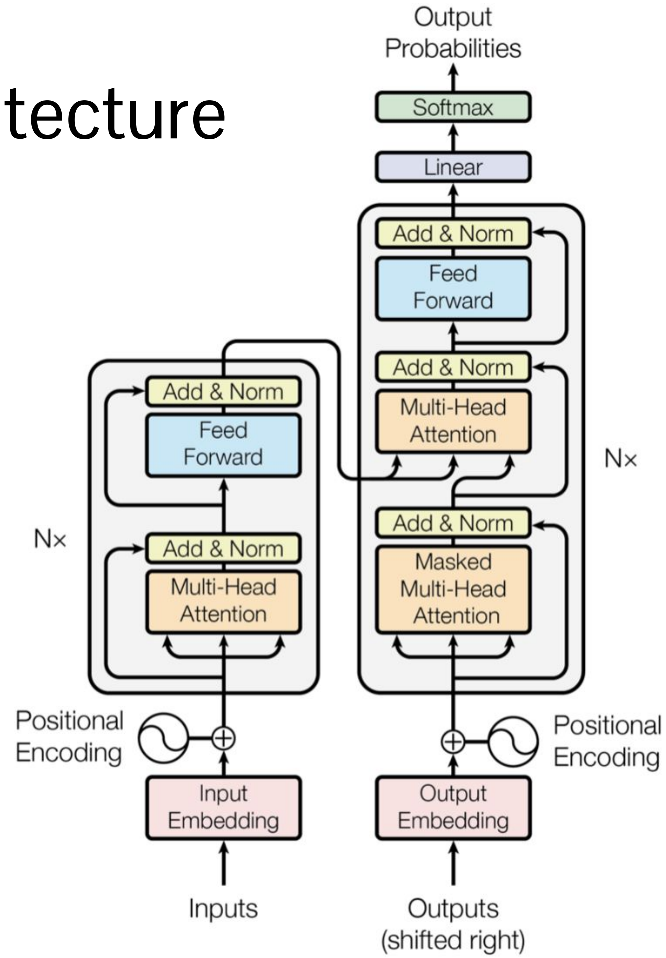


Figure 1: The Transformer - model architecture.

Applications

- Machine translation

Applications

- Text generation

SYSTEM PROMPT (HUMAN-WRITTEN)

Miley Cyrus was caught shoplifting from Abercrombie and Fitch on Hollywood Boulevard today.

MODEL COMPLETION (MACHINE-WRITTEN, SECOND TRY)

The 19-year-old singer was caught on camera being escorted out of the store by security guards.

The singer was wearing a black hoodie with the label ‘Blurred Lines’ on the front and ‘Fashion Police’ on the back.

Scroll down for video

Shoplifting: Miley Cyrus was caught shoplifting from Abercrombie and Fitch on Hollywood Boulevard today (pictured)

The singer was wearing a black hoodie with the label ‘Blurred Lines’ on the front and ‘Fashion Police’ on the back

The singer was also wearing a pair of black-rimmed glasses, a black jacket, black jeans and black sandals.

She was carrying a pair of black and white striped gloves and a small black bag.

https://openai.com/blog/better-language-models/

Applications

- Music sequence generation

https://openai.com/blog/musenet/

Next

- Variational Autoencoder