Machine Translation (Part I)

Peerachet Porkaew, NECTEC

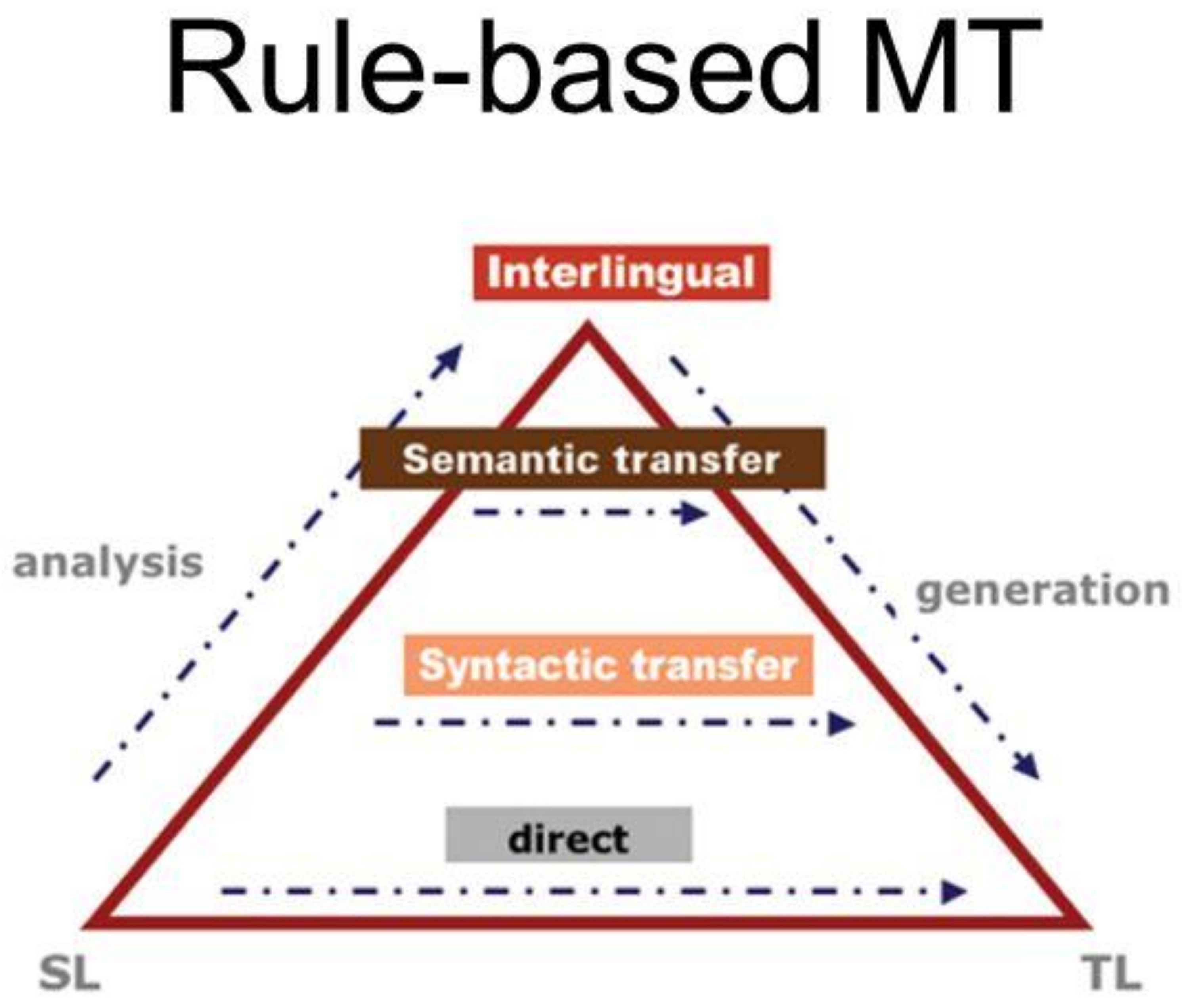
Outline

- Rule-based Machine Translation
- Statistical Machine Translation (SMT)
- Neural Machine Translation (NMT)
  - Sequence-to-Sequence
  - RNN-based NMT + Attention
  - Transformer

Rule-based Machine Translation

Rule-based Machine Translation (RBMT)

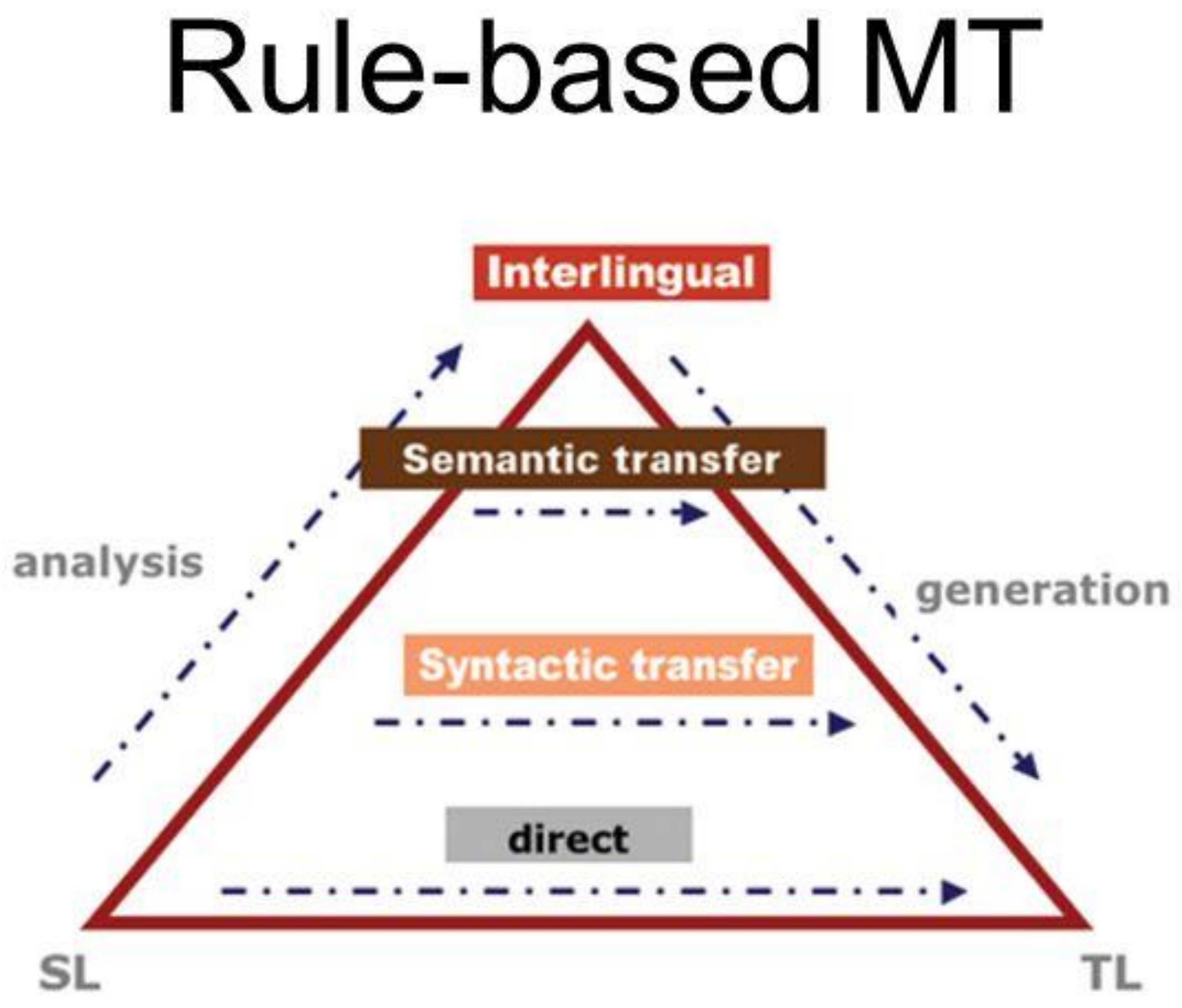
- Linguistic Knowledge
- Three sub-module :
  - Analysis
  - Transfer
  - Generation
- Template-based Translation



The Vauquois Triangle

Rule-based Machine Translation (RBMT)

- Limitations :
  - Not Automatic
  - Time consuming
  - Conflicts
  - Less Clarity



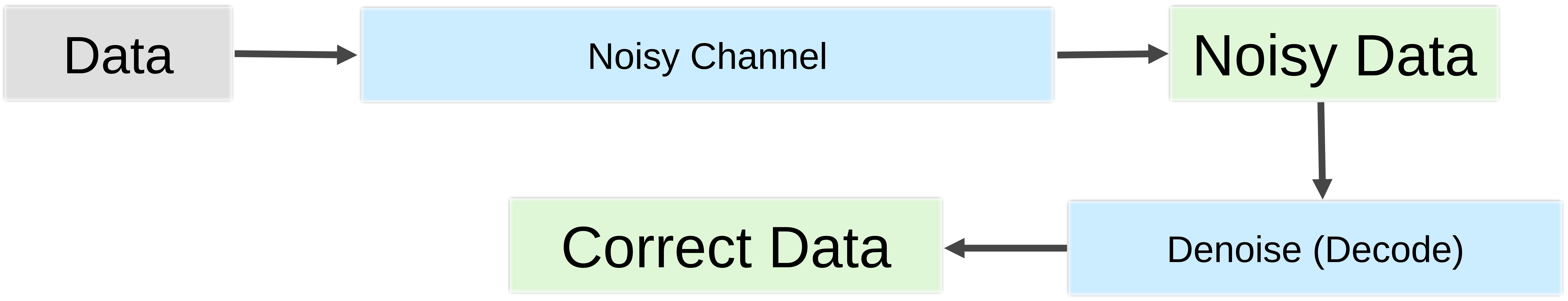
The Vauquois Triangle

Simple RBMT Demo

Statistical Machine Translation

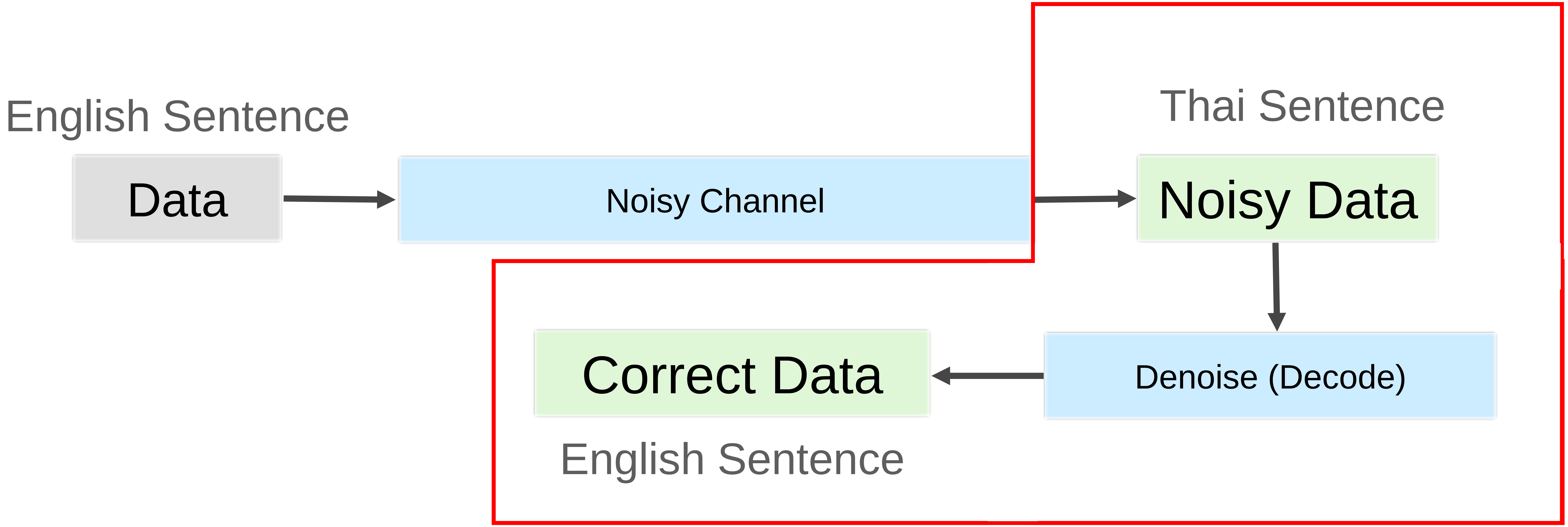
Statistical Machine Translation

- Noisy Channel Model



Statistical Machine Translation

- Noisy Channel Model



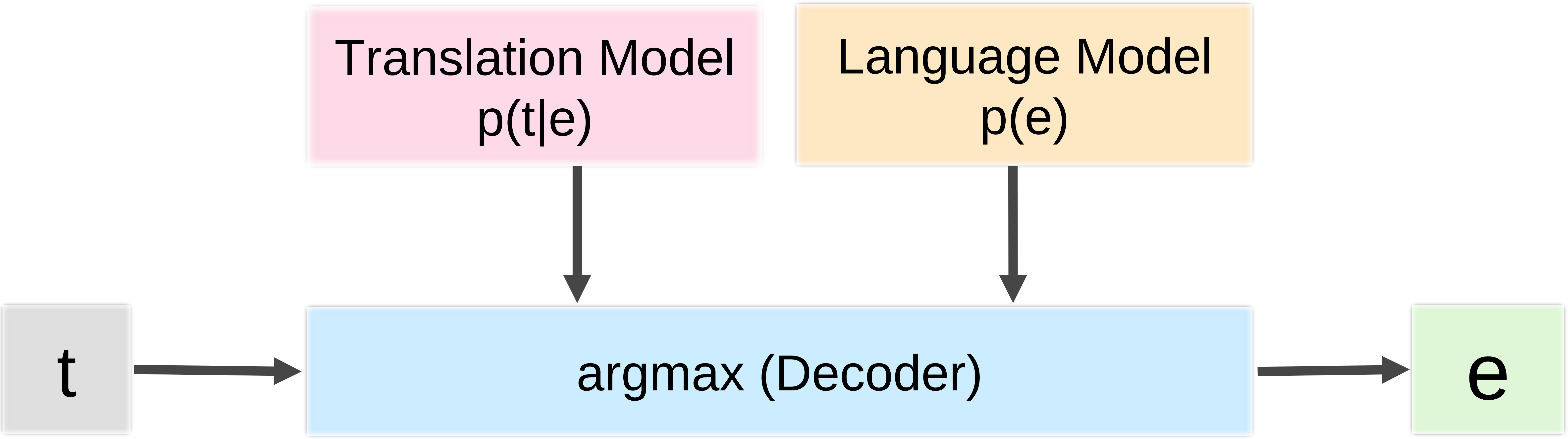
Statistical Machine Translation

- Source sentence ***t***, e.g. Thai
- Target sentence ***e***, e.g. English
- Probabilistic formulation using Bayes rule $P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$

$$\hat{e} = \mathrm{argmax}_e\, \mathrm{p}(e|t)$$
$$\hat{e} = \mathrm{argmax}_e\, \mathrm{p}(t|e)p(e)$$

Statistical Machine Translation (Cont.)

$$\hat{e} = \mathrm{argmax}_e\, \mathrm{p}(t|e)p(e)$$



Statistical Decoder (Simplified version)

เมื่อวาน	นี้	ฉัน	ไป	ทะเล	กับ	เพื่อน
Yesterday	this	I	go	the sea	with	friends
Yesterday		I	went to	sea	with	friend
Previous day		me	get to	ocean	with my friend	

เมื่อวาน	นี้	ฉัน	ไป	ทะเล	กับ	เพื่อน
Yesterday	this	I	go	the sea	with	friends
Yesterday		I	went to	sea	with	friend
Previous day		me	get to	ocean	with my friend	

เมื่อวาน	นี้	ฉัน	ไป	ทะเล	กับ	เพื่อน
Yesterday	this	I	go	the sea	with	friends
Yesterday		I	went to	sea	with	friend
Previous day		me	get to	ocean	with my friend	

Translation Model *p*(t|e)

- Word-based Translation Model

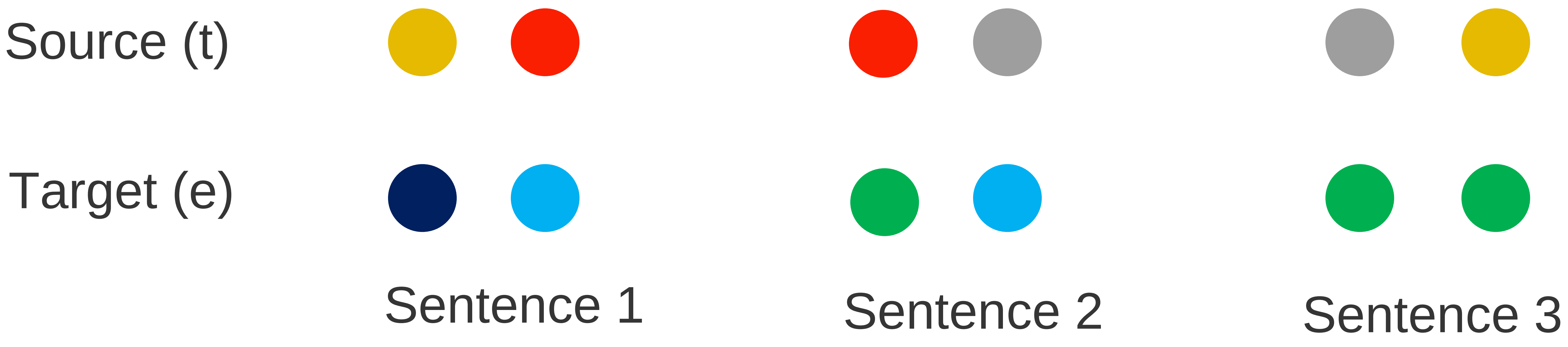
$$p(t|e) = p\left(\text{ไป}\middle|go\right) = 0.5$$

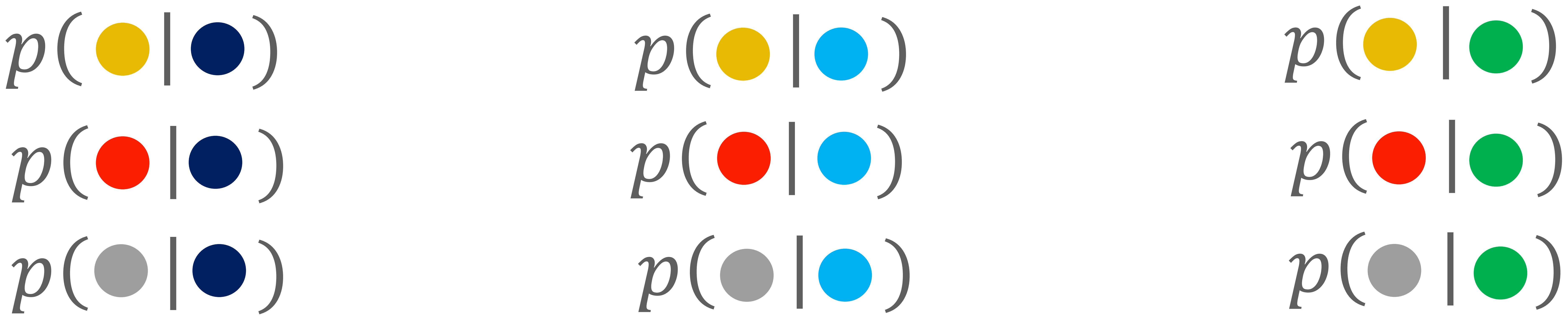
- Phrase-based Translation Model

$$\mathrm{p}(t_1 t_2, \ldots, t_n | e_1, e_2, \ldots, e_m) = p(\text{เมื่อวาน นี้}|\ \text{previous day}) = 0.1$$

How we get the *p*(t|e) ?

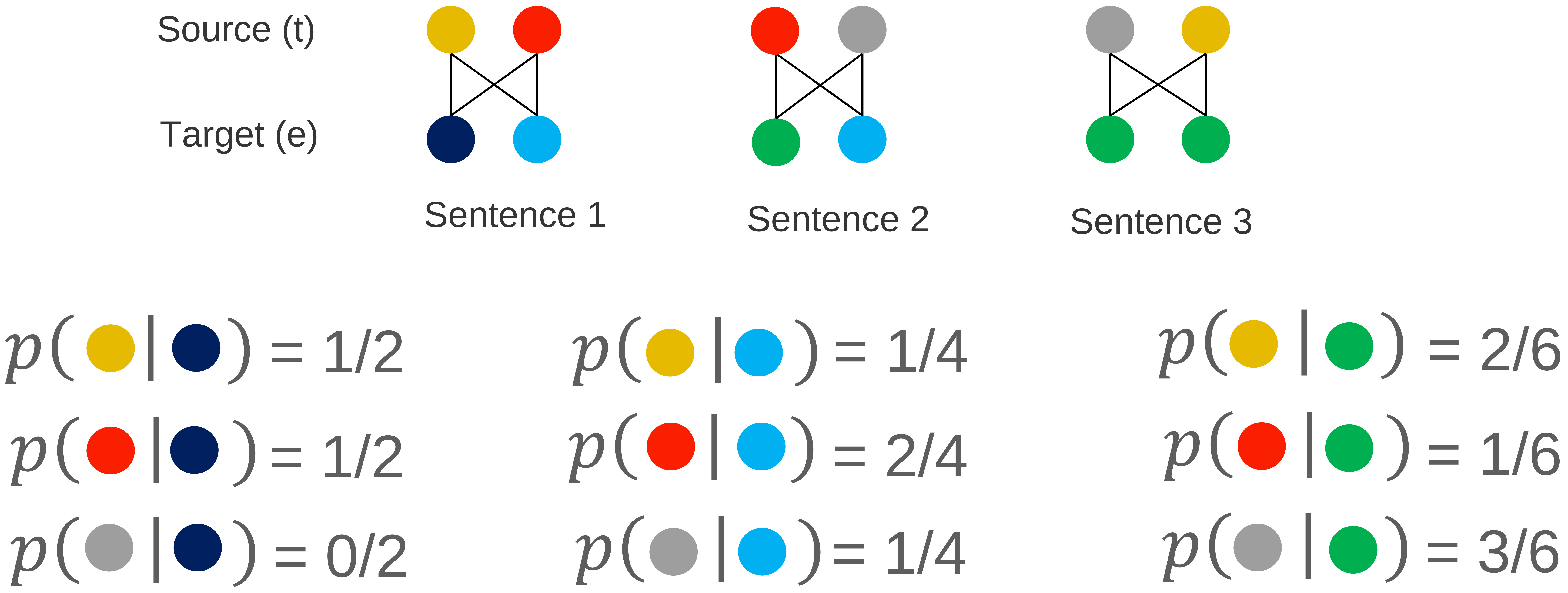
- We do not have alignments. No problem we assume alignment are uniformly paired. ***p(t|e) = c*** (constant)





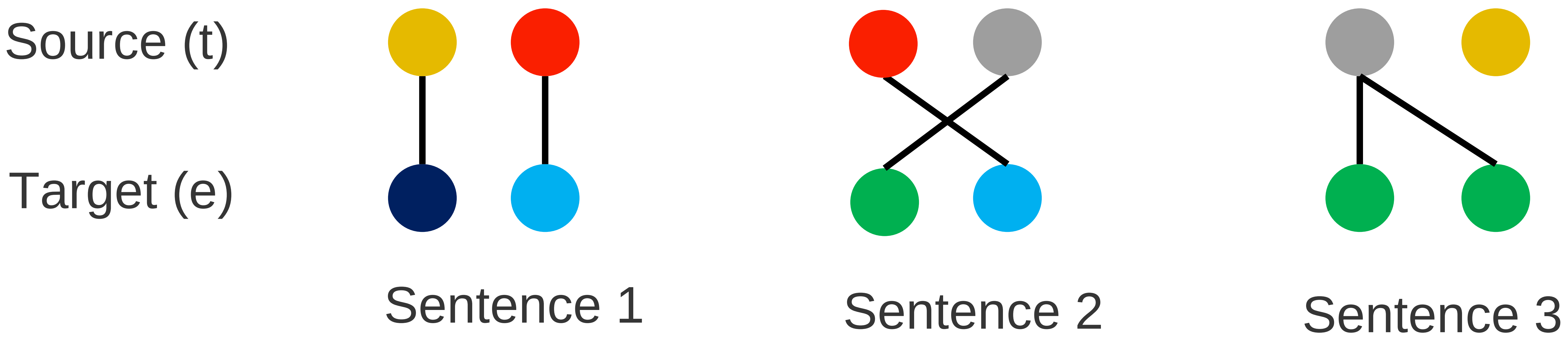
How we get the *p*(t|e) ?

- But, we do not have alignment. No problem we assume alignment are uniformly paired. ***p*(t|e) = *c*** (constant)



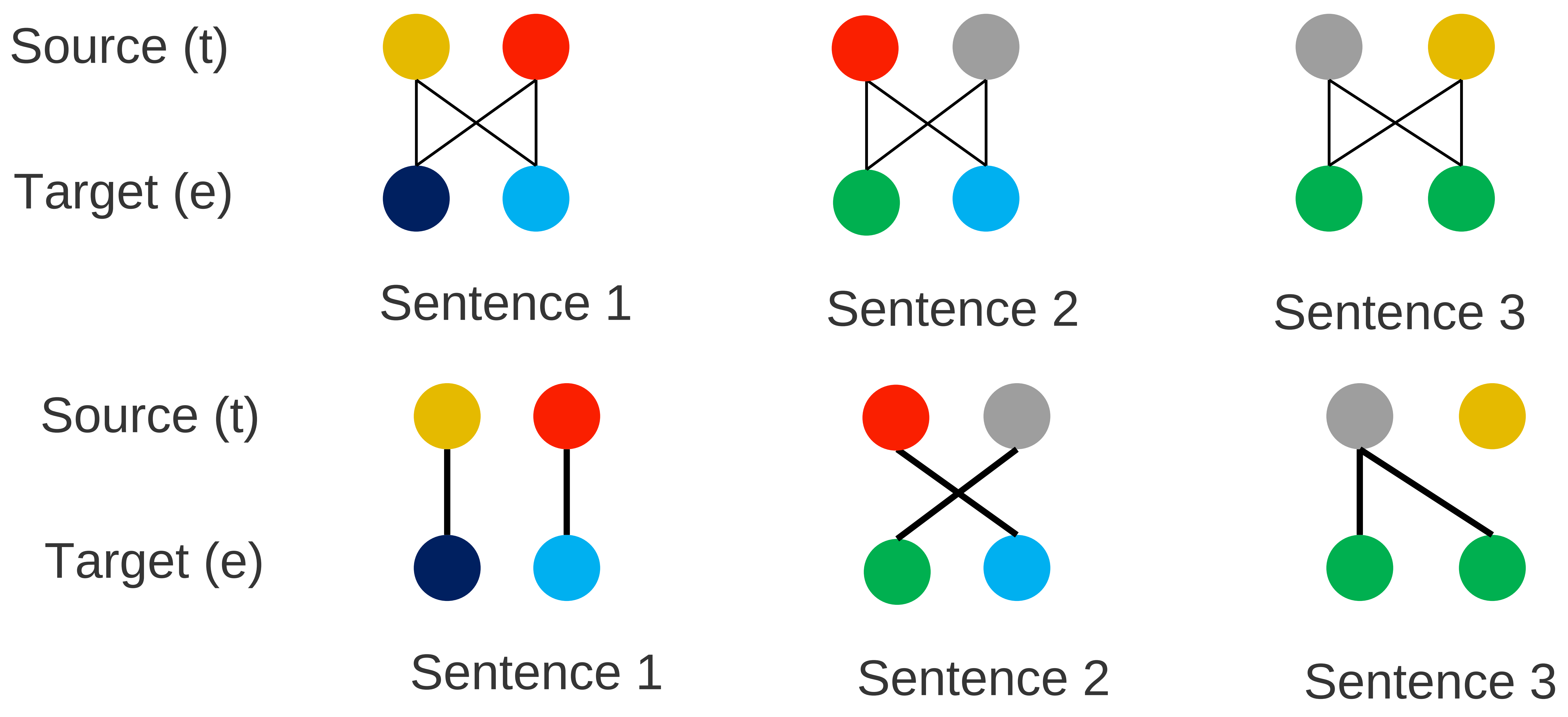
How we get the *p*(t|e) ?

- Now, we can get the better alignment from previous knowledge.



Then, we can calculate *p(t|e)* again using these alignment information.

How we get the *p*(t|e) ?



Expectation Maximization (EM)

Translation Model

- **Word-based Translation Model**

$$p(t|e) = p(\text{ไป}|go) = 0.5$$

- Phrase-based Translation Model

$$\mathrm{p}(t_1 t_2, \ldots, t_n | e_1, e_2, \ldots, e_m) = p(\text{เมื่อวาน นี้}|\ \text{previous day}) = 0.1$$

Word-based Translation Model

- **Word-based Translation Model**

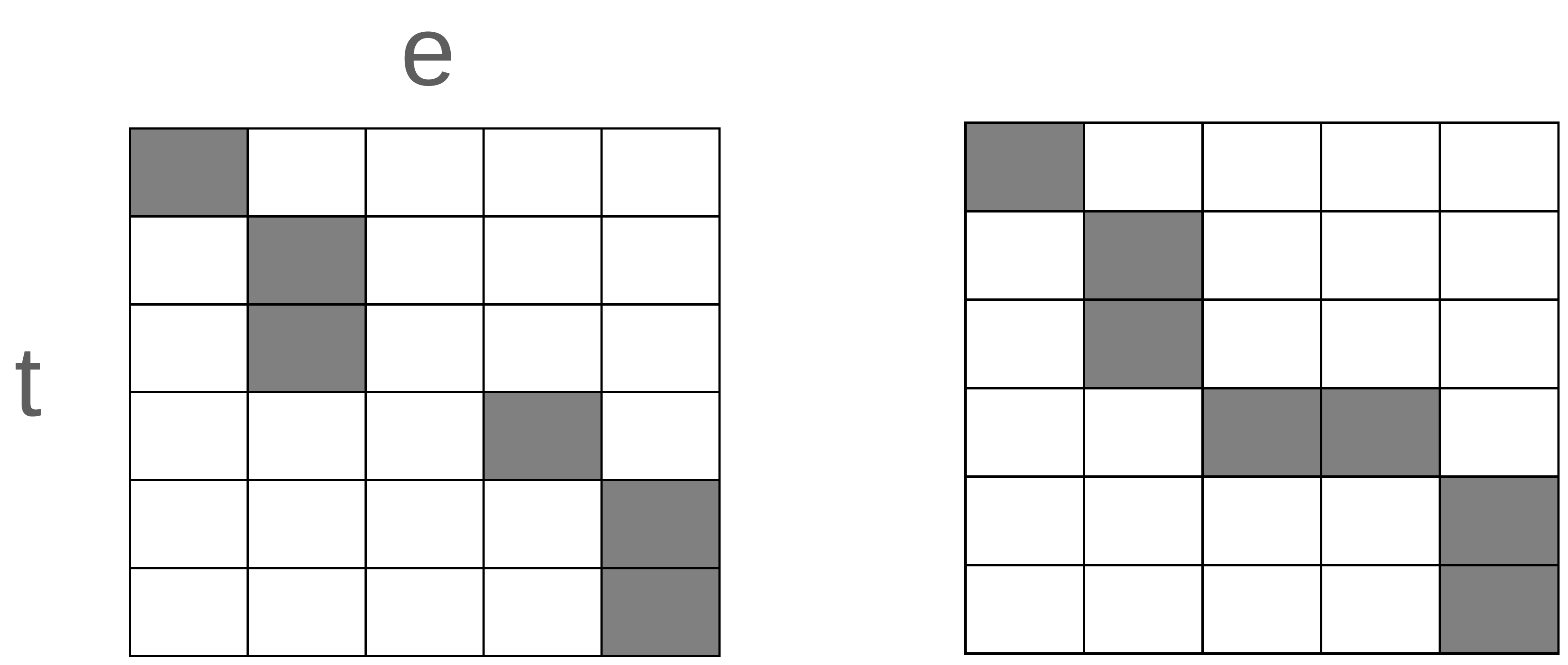
$$p(t|e) = p\left(\text{ไป}|go\right) = 0.5$$

- **Phrase-based Translation Model**

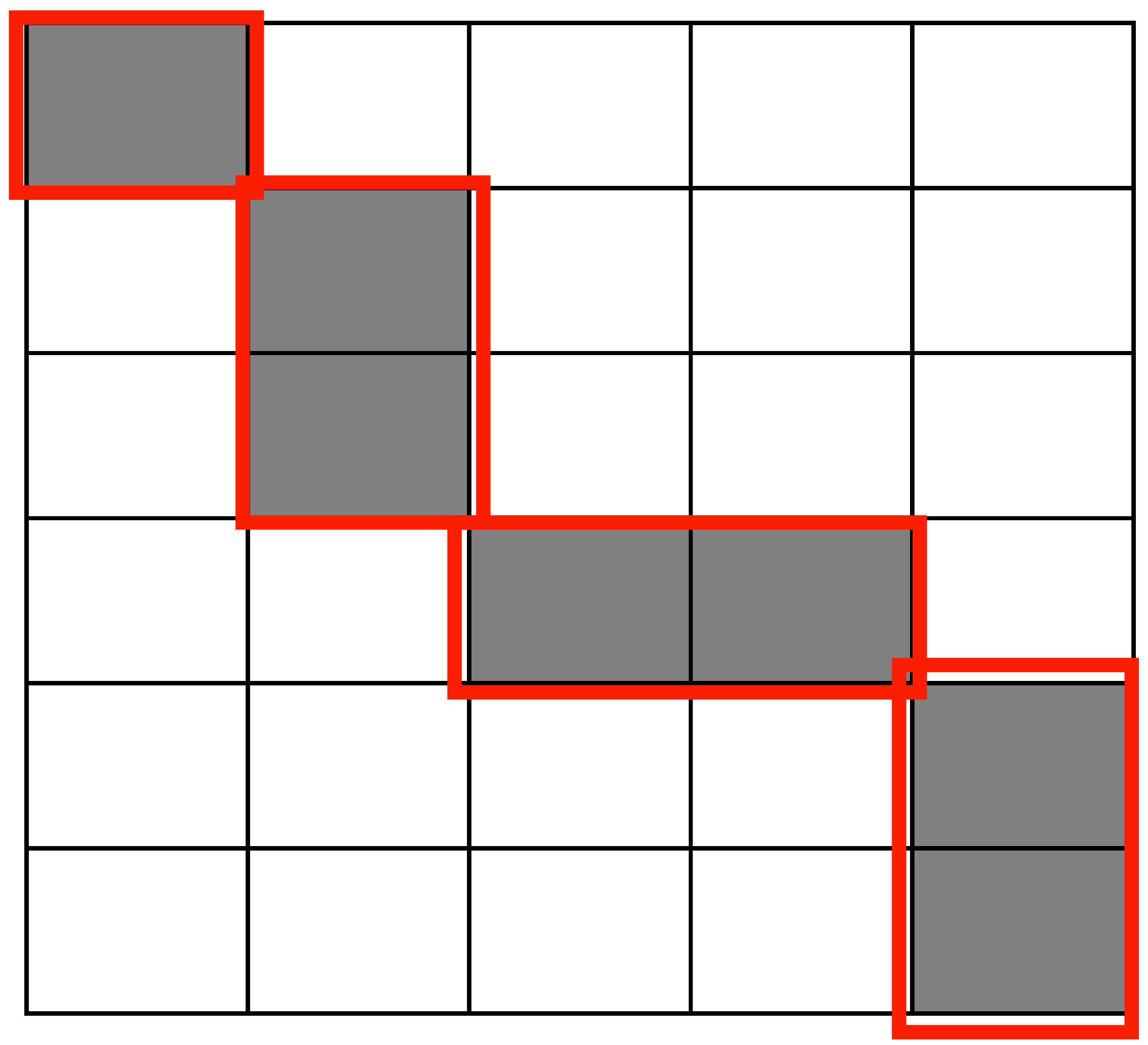
$$\mathrm{p}(t_1 t_2, \ldots, t_n | e_1, e_2, \ldots, e_m) = p(\text{เมื่อวาน นี้}|\ \text{previous day}) = 0.1$$

How we get the P(t|e) for phrases ?

- Phrase Extraction Algorithm
  - Expanding single word alignment pairs to multiple-word alignment.

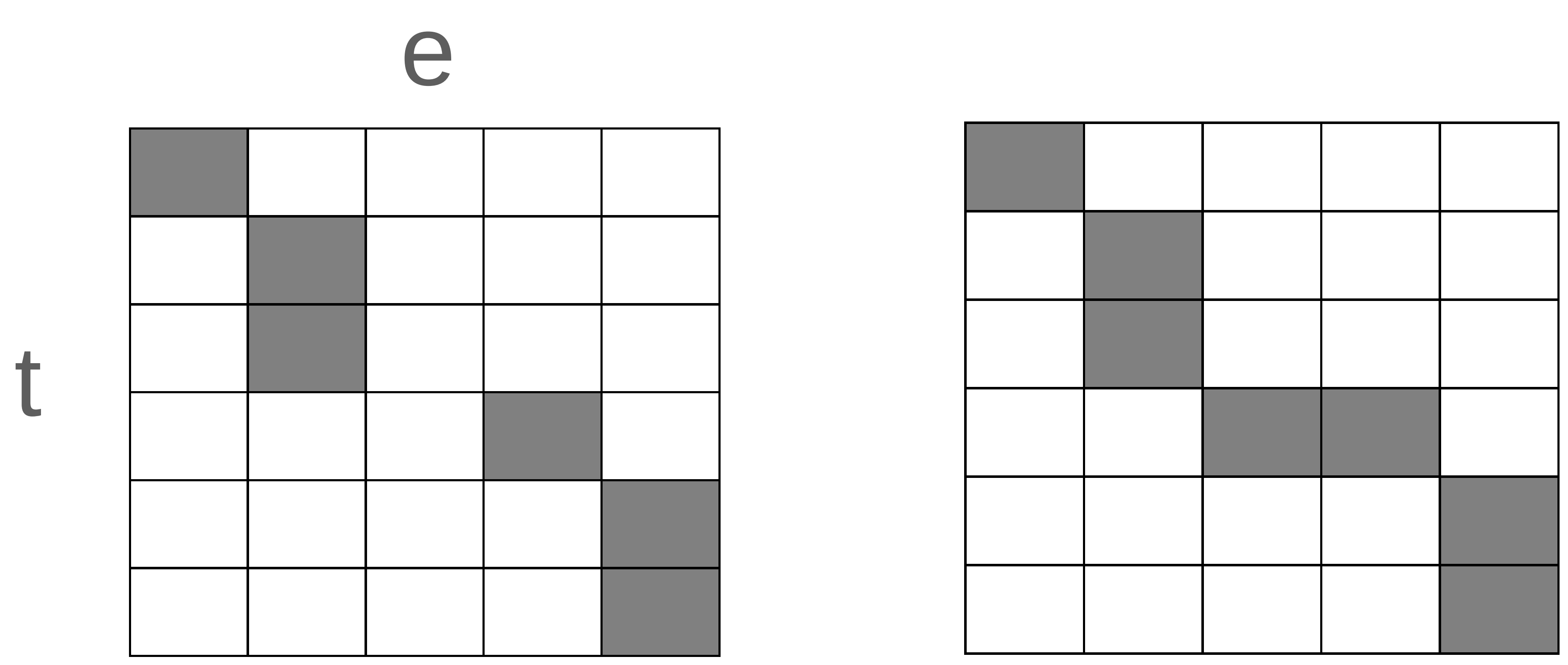


Fill missing alignment pairs

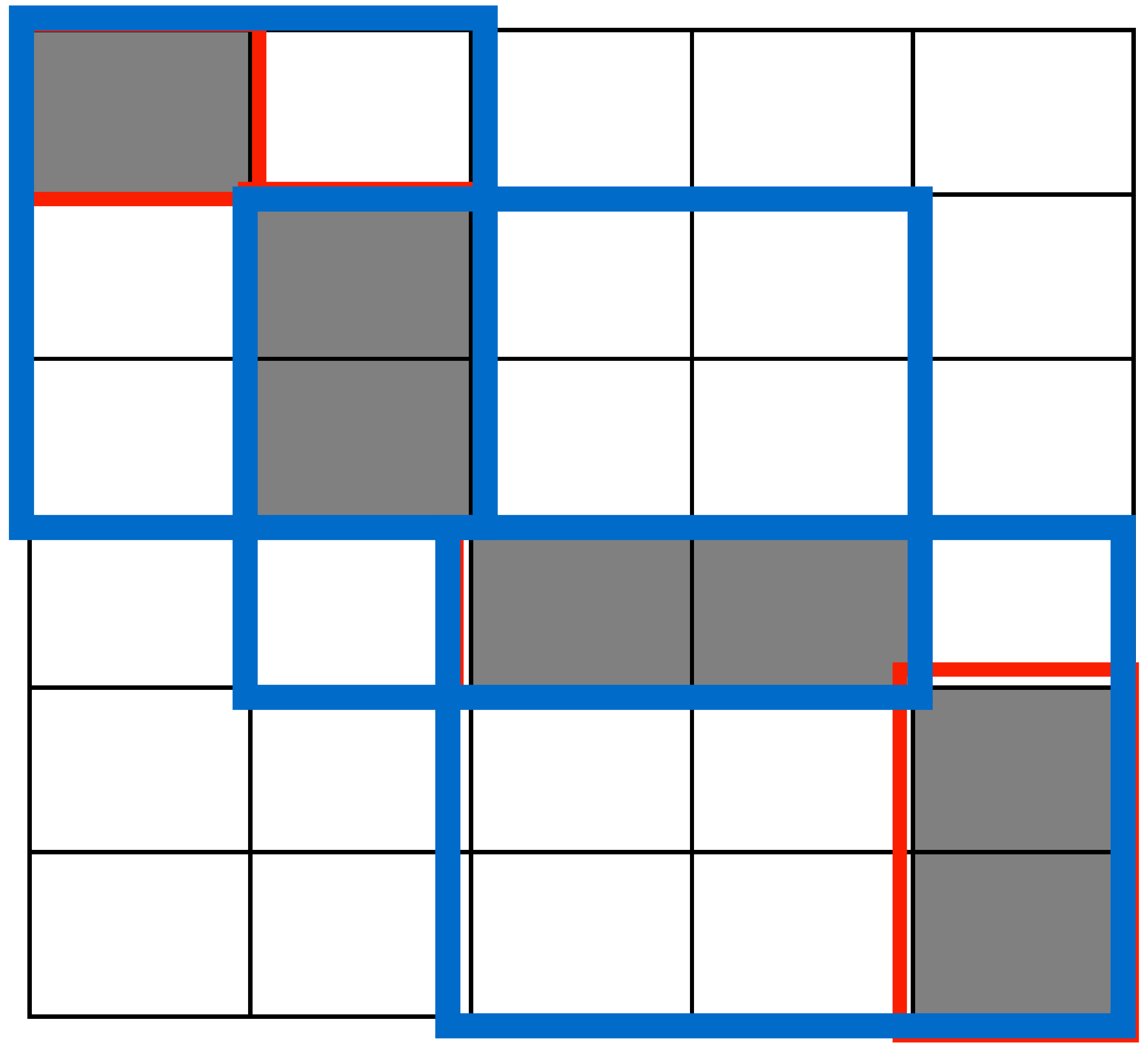


How we get the P(t|e) for phrases ?

- Phrase Extraction Algorithm
  - Expanding single word alignment pairs to multiple-word alignment.



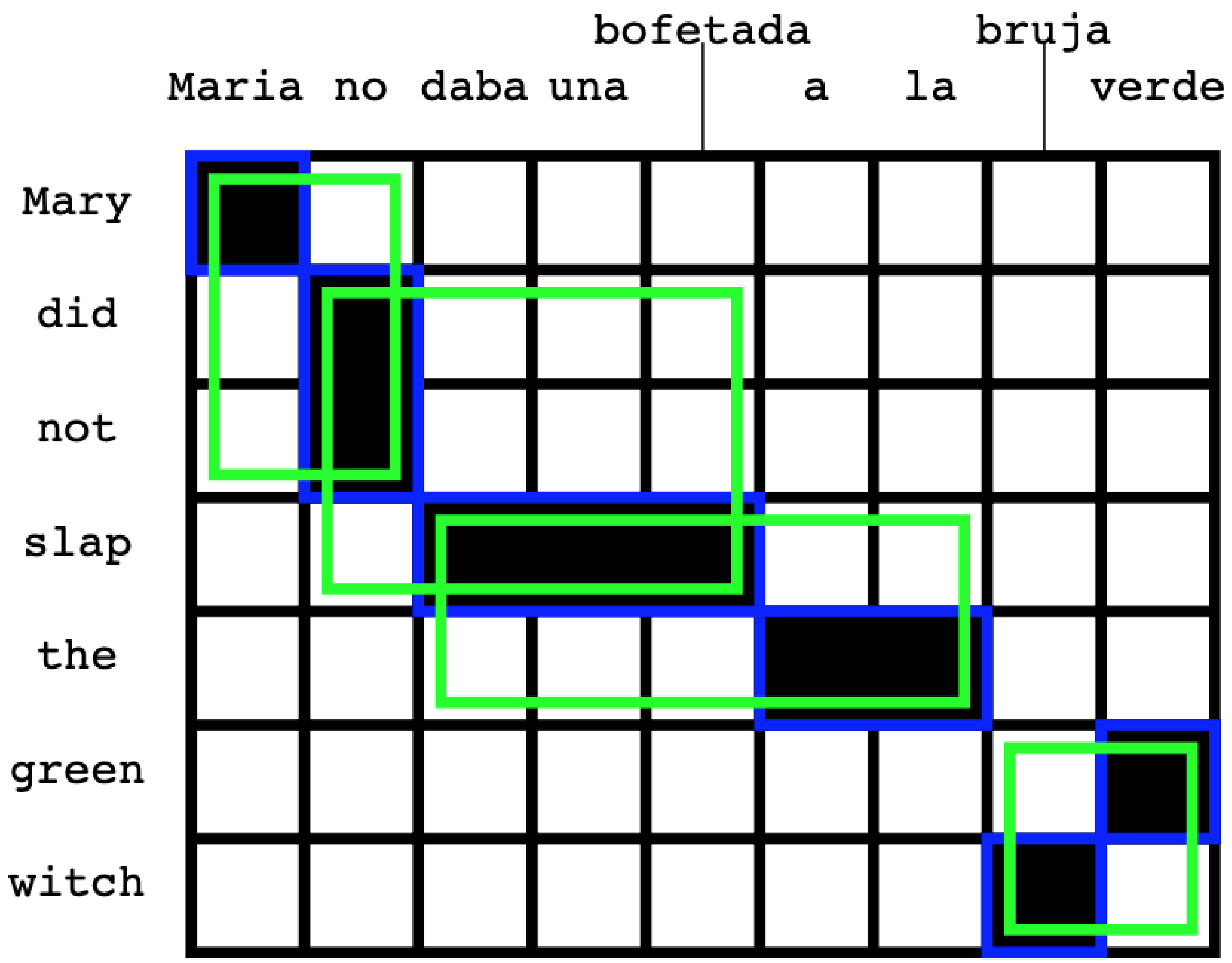
Fill missing alignment pairs



How we get the P(t|e) for phrases ?

- Phrase Pairs

$$p(t|e) = \frac{count(t,e)}{count(e)}$$



(Maria, Mary), (no, did not), (slap, daba una bofetada), (a la, the), (bruja, witch), (verde, green), (Maria no, Mary did not), (no daba una bofetada, did not slap), (daba una bofetada a la, slap the), (bruja verde, green witch)

Image from : www.statmt.org

Statistical Decoder (with Translation model)

เมื่อวาน	นี้	ฉัน	ไป	ทะเล	กับ	เพื่อน
Yesterday (0.5)	this (1.0)	I (0.8)	go (0.6)	the sea (0.7)	with (0.8)	friends (0.6)
Yesterday (0.7)		I (0.8)	went to (0.3)	sea (0.2)	with (0.8)	friend (0.4)
Previous day (0.1)		me (0.2)	get to (0.01)	ocean (0.05)	with my friend (0.1)	

Language Model $p_{LM}(e)$

- Language Model of the target language.
- Calculate the **“*fluency*”** of sentences

$$p_{LM}(\text{I went to the sea}) \;>\; p_{LM}(\text{I went to ocean})$$

How can we estimate p_{LM} ?

- **N-gram** language model

1-gram : p_{LM}(I went to the sea) =
p(I) x p(went) x p(to) x p(the) x p(sea)

2-gram : p_{LM}(I went to the sea) =

p(I |<bos>) x p(went| I) x p(to|went) x p(the|to) x p(sea|the)

How can we estimate P_{LM} ?

- **Maximum Likelihood Estimation (MLE)**

- p(I) = count(“I”) / N

- p(went|I) = count(“I went”) / count(“I”)

Smoothing

- if p(x|y) = 0 ? ➔ $P_{LM} = 0$

We can use smoothing technique to overcome this situation.

For example, **Add-one smoothing (Laplace Smoothing)**

$$P^*_{\text{Laplace}}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{\sum_w (C(w_{n-1}w)+1)} = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V}$$

Smoothing

- **Back-off**

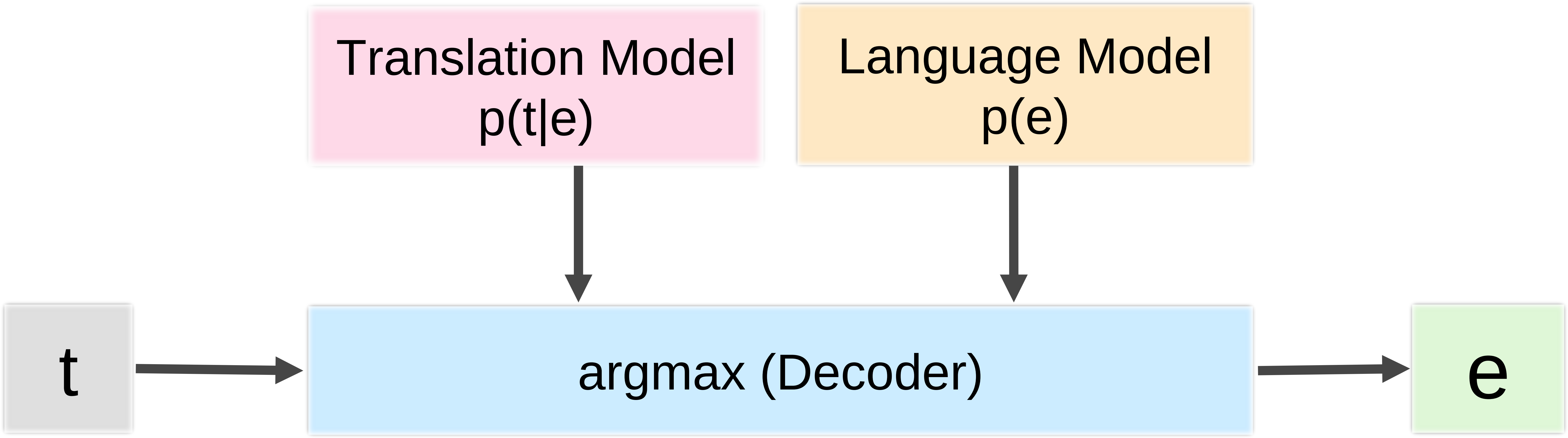
$$\text{if } p(w_i|w_{i-1}) = 0 \text{ then } \hat{p}(w_i|w_{i-1}) = p(w_i)$$

- **Interpolation**

$$\hat{p}(w_i|w_{i-1}) = \lambda_1 p(w_i) + \lambda_2 p(w_i|w_{i-1})$$
$$\lambda_1 + \lambda_2 = 1$$

Statistical Decoder

$$\hat{e} = \mathrm{argmax}_e\, p(t|e)p(e)$$



Decoding

- Decoding with Translation Model and Language Model

<table>
<tr><th>เมื่อวาน</th><th>นี้</th><th>ฉัน</th><th>ไป</th><th>ทะเล</th><th>กับ</th><th>เพื่อน</th></tr>
<tr><td>Yesterday
(0.5)</td><td>this
(1.0)</td><td>I
(0.8)</td><td>go
(0.6)</td><td>the sea
(0.7)</td><td>with
(0.8)</td><td>friends
(0.6)</td></tr>
<tr><td colspan="2">Yesterday
(0.7)</td><td>I
(0.8)</td><td>went to
(0.3)</td><td>sea
(0.2)</td><td>with
(0.8)</td><td>friend
(0.4)</td></tr>
<tr><td colspan="2">Previous day
(0.1)</td><td>me
(0.2)</td><td>get to
(0.01)</td><td>ocean
(0.05)</td><td colspan="2">with my friend
(0.1)</td></tr>
</table>

Score = 0.7 x 0.8 x 0.6 x 0.7 x 0.8 x 0.6 x P_{LM}(“Yesterday I go the sea with friends”)

Decoding

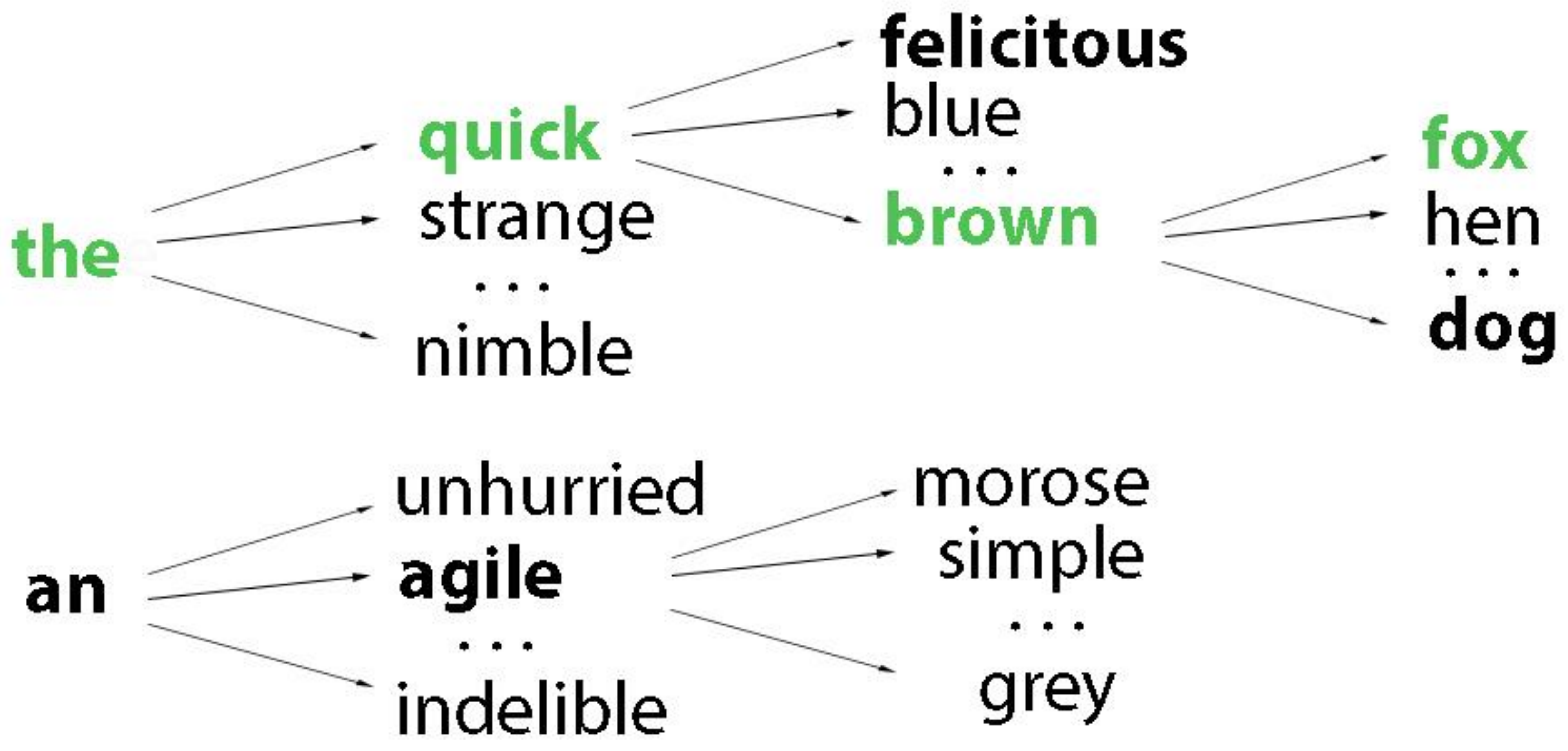
- Decoding with Translation Model and Language Model

เมื่อวาน	นี้	ฉัน	ไป	ทะเล	กับ	เพื่อน
Yesterday (0.5)	this (1.0)	I (0.8)	go (0.6)	the sea (0.7)	with (0.8)	friends (0.6)
Yesterday (0.7)		I (0.8)	went to (0.3)	sea (0.2)	with (0.8)	friend (0.4)
Previous day (0.1)		me (0.2)	get to (0.01)	ocean (0.05)	with my friend (0.1)	

Score = 0.1 x 0.2 x 0.3 x 0.05 x 0.1x P_{LM}(“Previous day me went to ocean with my friend”)

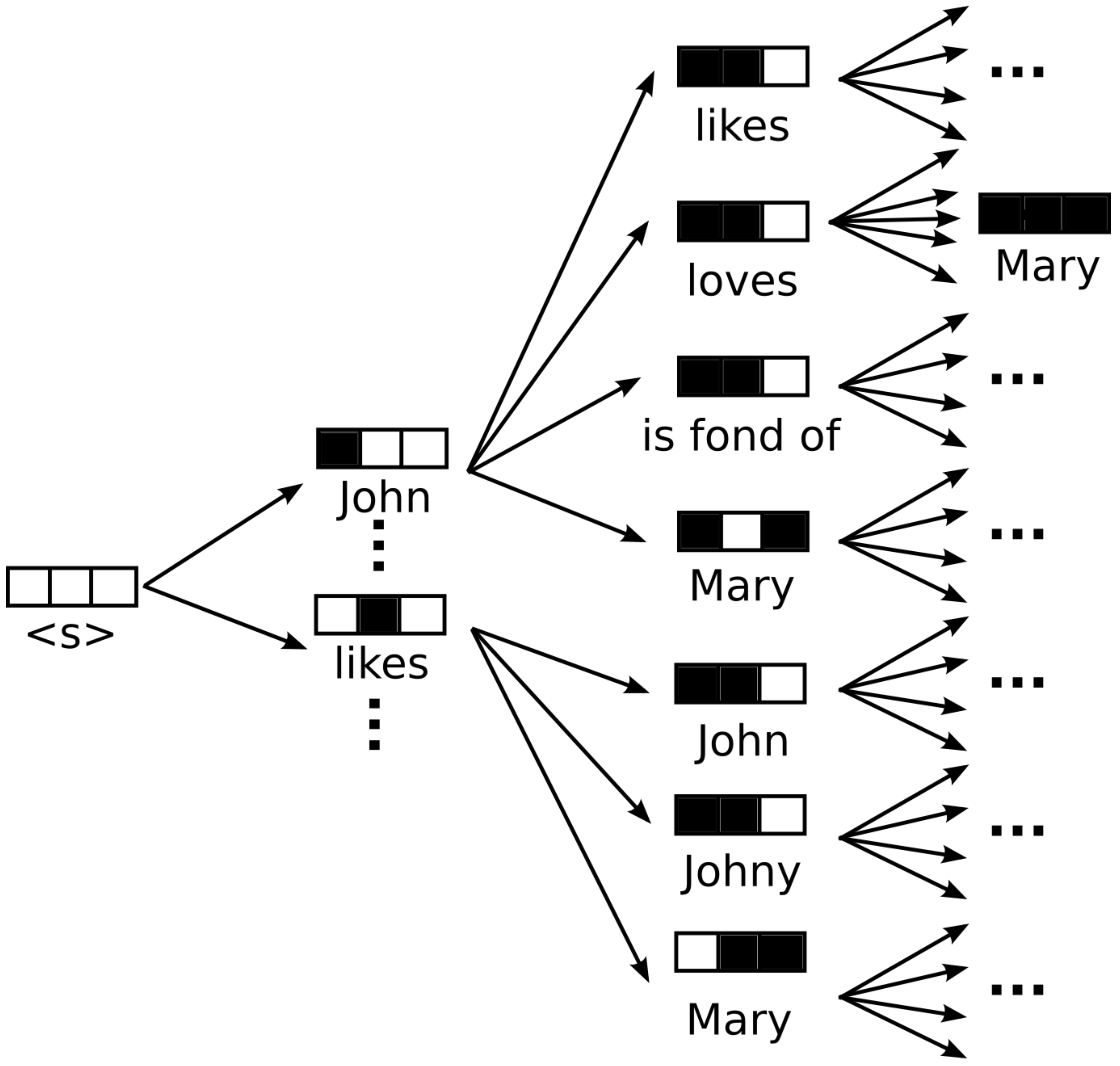
Beam Search

The Beam Search is a tree search algorithm but the data are filtered and sorted using a heuristic function.



Left-to-Right Beam Search

Beam Search



Beam Search in Statistical Decoder

SMT using

NLTK Library

main_smt.py

```python
from nltk.translate import PhraseTable, StackDecoder
from collections import defaultdict
from math import log

def test_smt():
    phrase_table = PhraseTable()
    phrase_table.add(('book',), ('หนังสือ',), log(0.1))
    phrase_table.add(('this','book',), ('หนังสือ','เล่ม','นี้',), log(0.8))
    phrase_table.add(('this',), ('นี้',), log(0.8))
    phrase_table.add(('costs',), ('ราคา',), log(0.1))
    phrase_table.add(('300',), ('300',), log(0.1))
    phrase_table.add(('300',), ('สาม','ร้อย',), log(0.5))
    phrase_table.add(('baht',), ('บาท',), log(0.8))

    language_prob = defaultdict(lambda: -999.0)
    language_prob[('เล่ม',)] = log(0.5)
    language_prob[('หนังสือ',)] = log(0.4)
    language_prob[('หนังสือ', 'เล่ม', 'นี้')] = log(0.7)
    language_prob[('บาท',)] = log(0.1)
    language_prob[('สาม','ร้อย',)] = log(0.7)
    language_model = type('',(object,),
                          {'probability_change': lambda self, context, phrase: language_prob[ph
                           'probability': lambda self, phrase: language_prob[phrase]})()

    stack_decoder = StackDecoder(phrase_table, language_model)

    out = stack_decoder.translate("this book costs 300 baht".split())
    print(out)
```

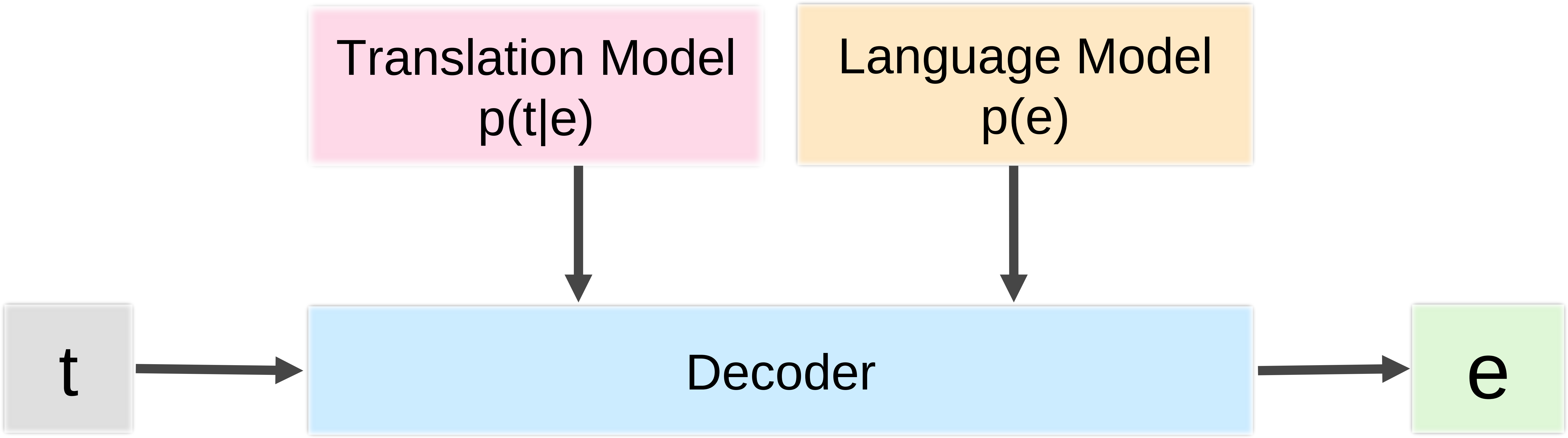
TERMINAL

```
['หนังสือ', 'เล่ม', 'นี้', 'ราคา', 'สาม', 'ร้อย', 'บาท']
```

REF: https://www.nltk.org/api/nltk.translate.stack_decoder.html

Summary

- Statistical Machine Translation
- Translation Model
- Language Model
- Decoder



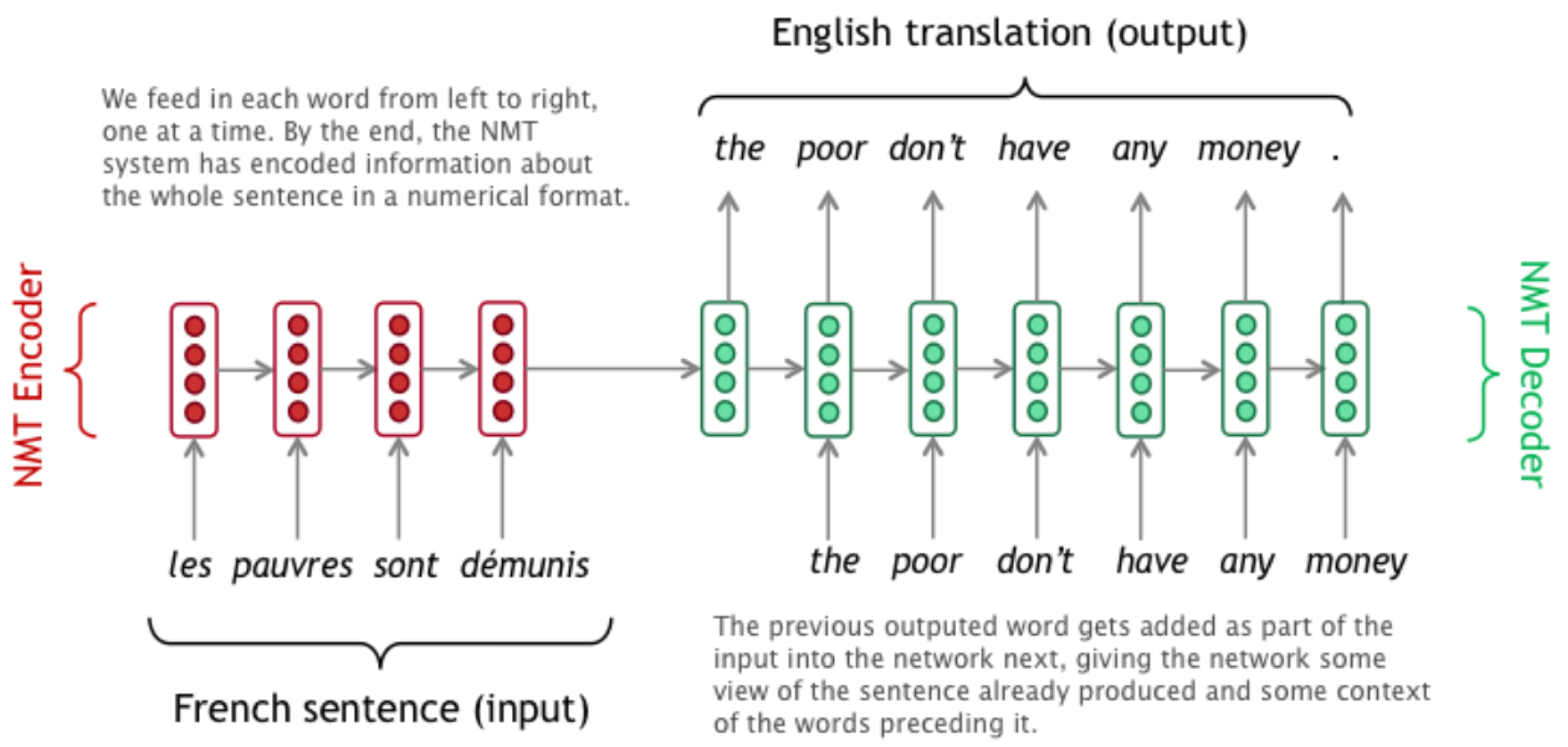
SMT Demo

Machine Translation (Part II)

Neural Machine Translation

Neural Machine Translation

- Deep Learning
- End-to-end training
- No word or phrase translation tables required



Background

- Recurrent Neural Network (RNN)
- Language model using long short-term memory (LSTM)
- Negative Log Likelihood (NLL)
- Seq2Seq model

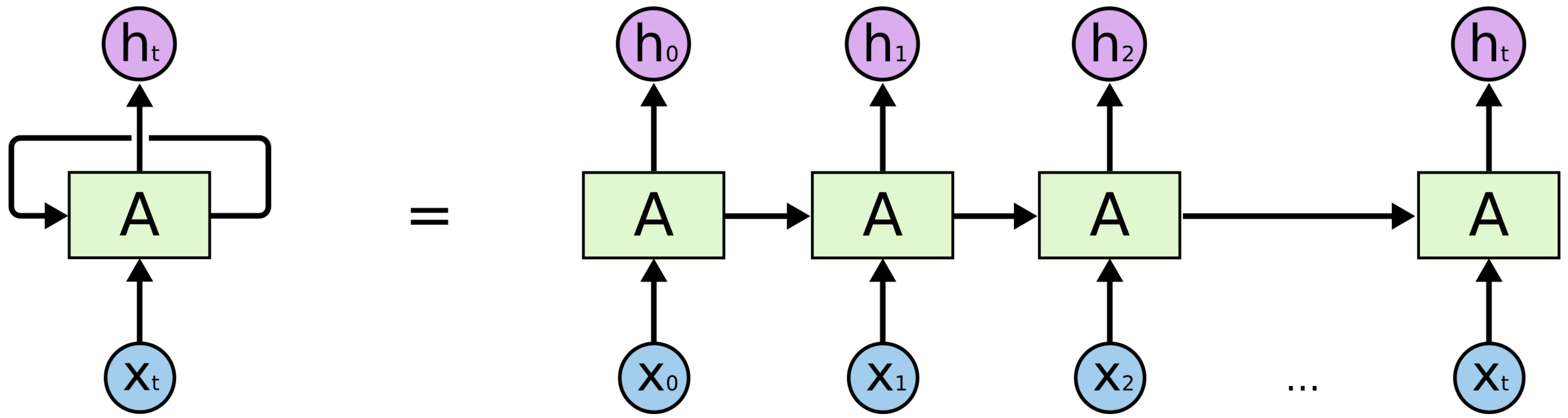
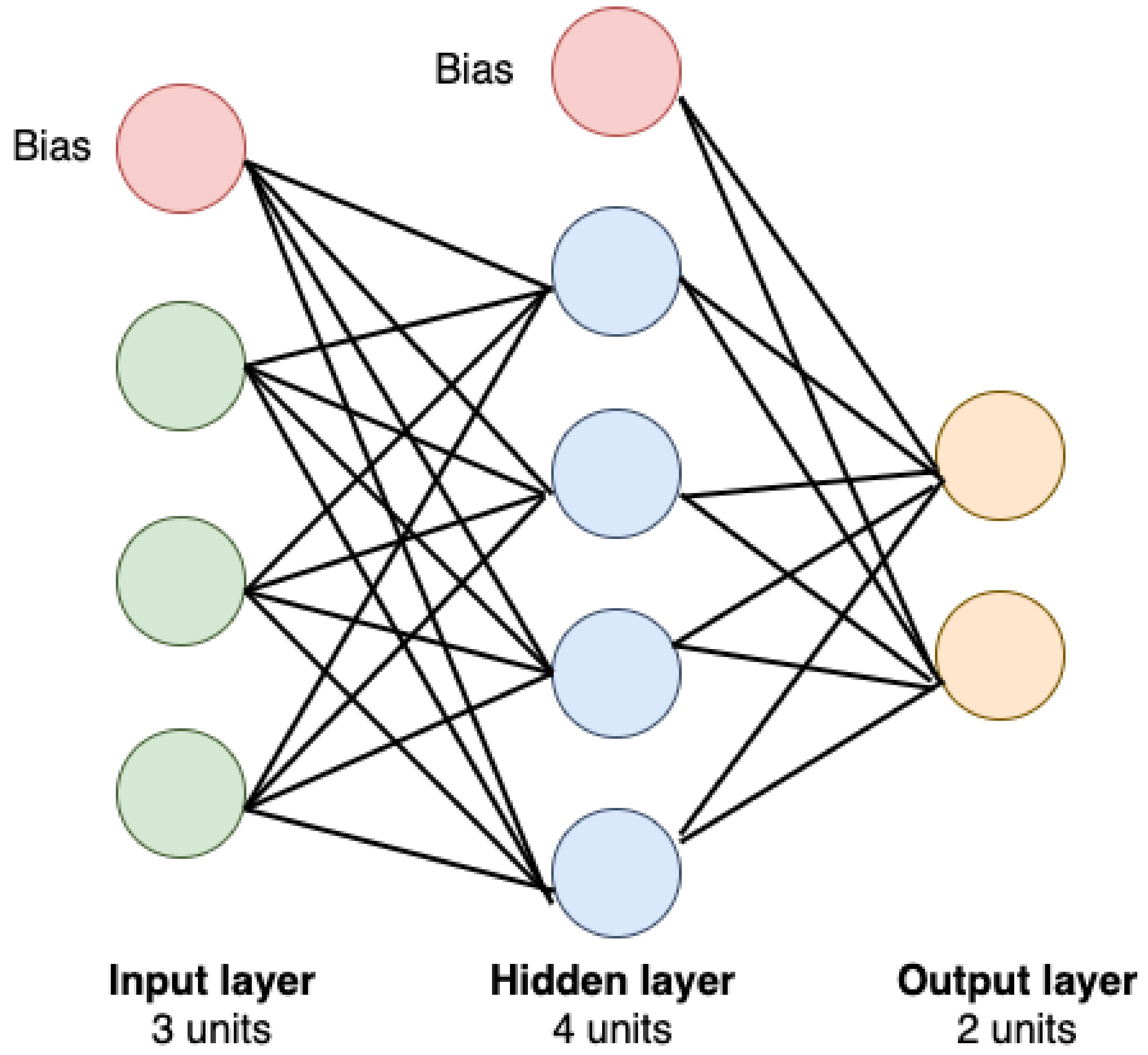
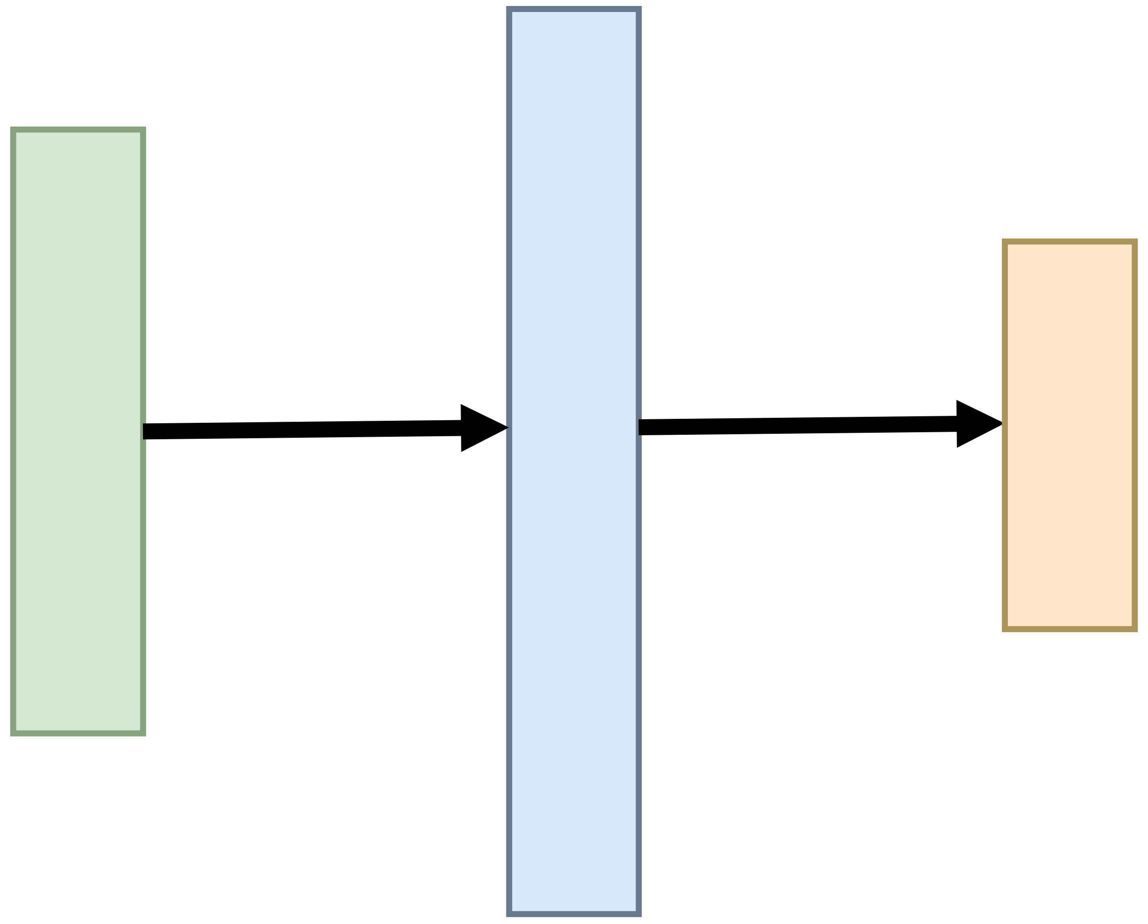


Image courtesy of Chris Olah

Neural Network

- Feed Forward Neural Network (FFN)



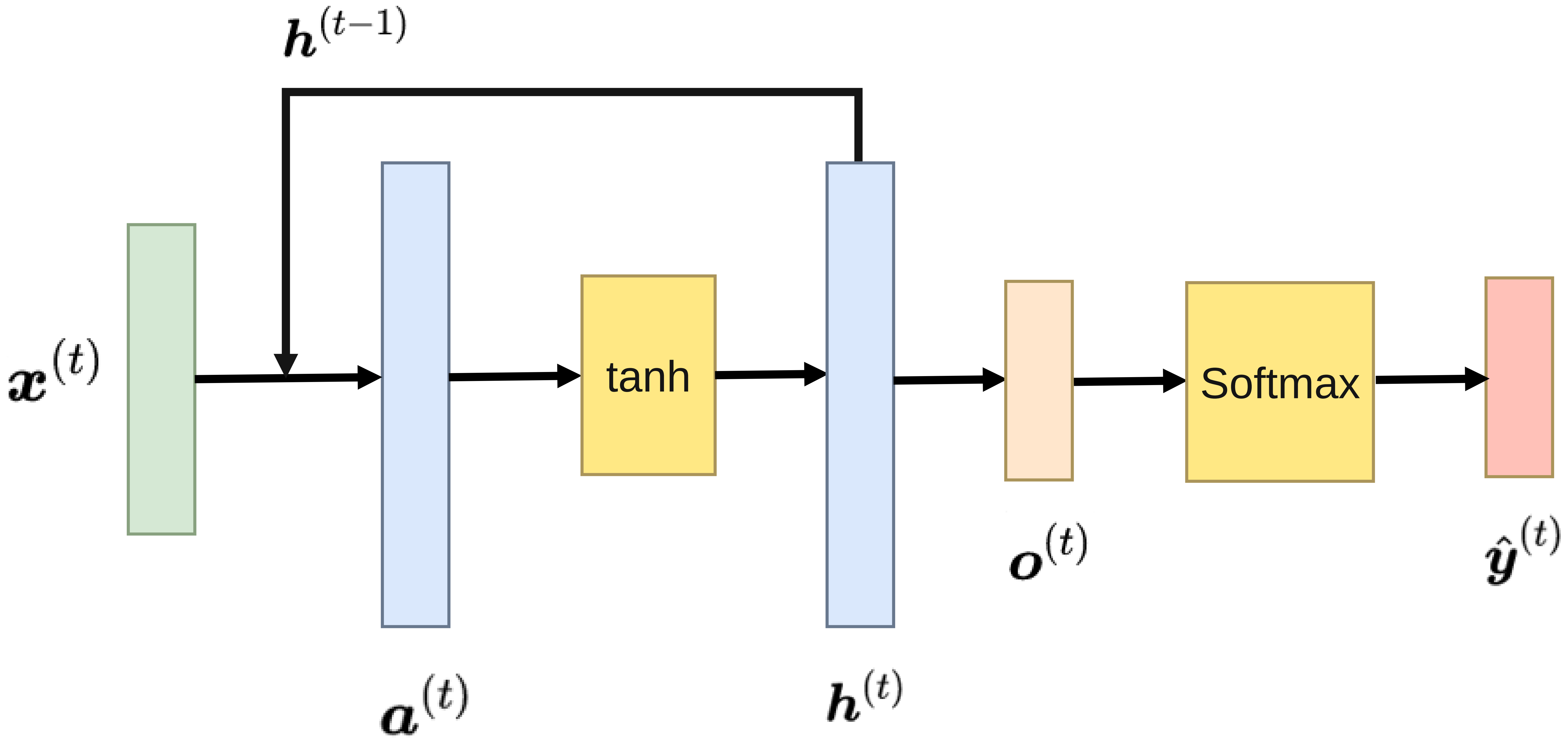


Simplified version of feed forward neural network

Image taken from https://miro.medium.com/

Recurrent Neural Network

- Recurrent Neural Network (RNN)



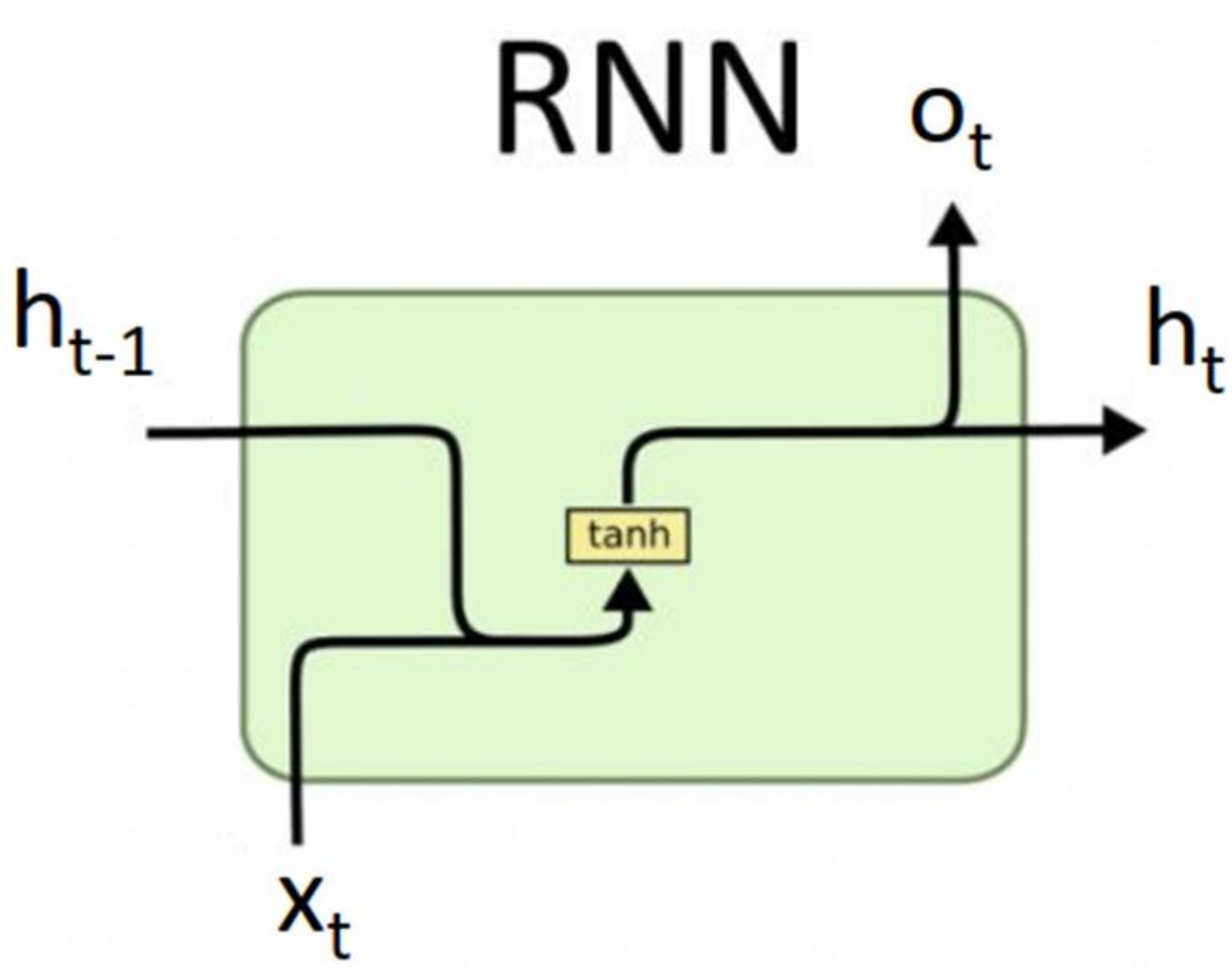
$$\boldsymbol{a}^{(t)} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}$$
$$\boldsymbol{h}^{(t)} = \tanh\left(\boldsymbol{a}^{(t)}\right)$$
$$\boldsymbol{o}^{(t)} = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)}$$
$$\hat{\boldsymbol{y}}^{(t)} = \operatorname{softmax}\left(\boldsymbol{o}^{(t)}\right)$$

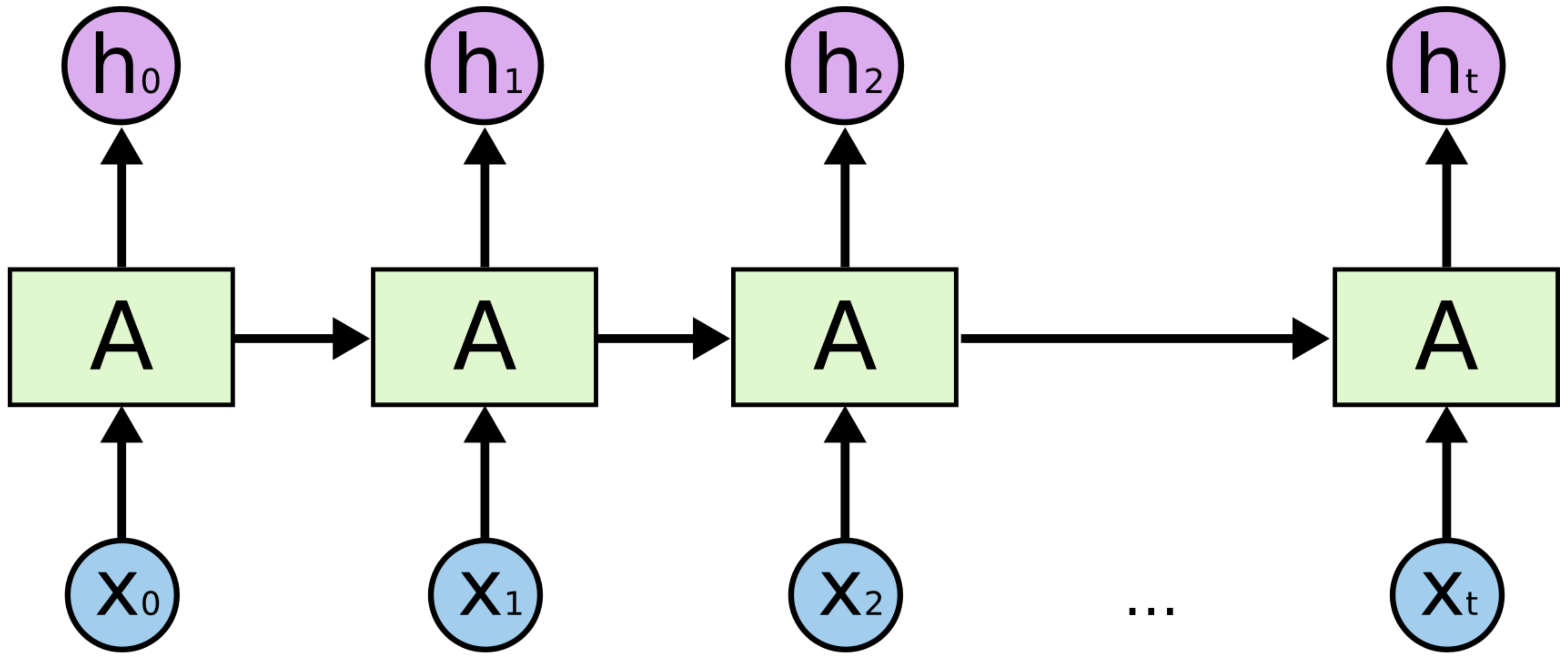
Softmax Function

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Recurrent Neural Network

- Recurrent Neural Network (RNN)

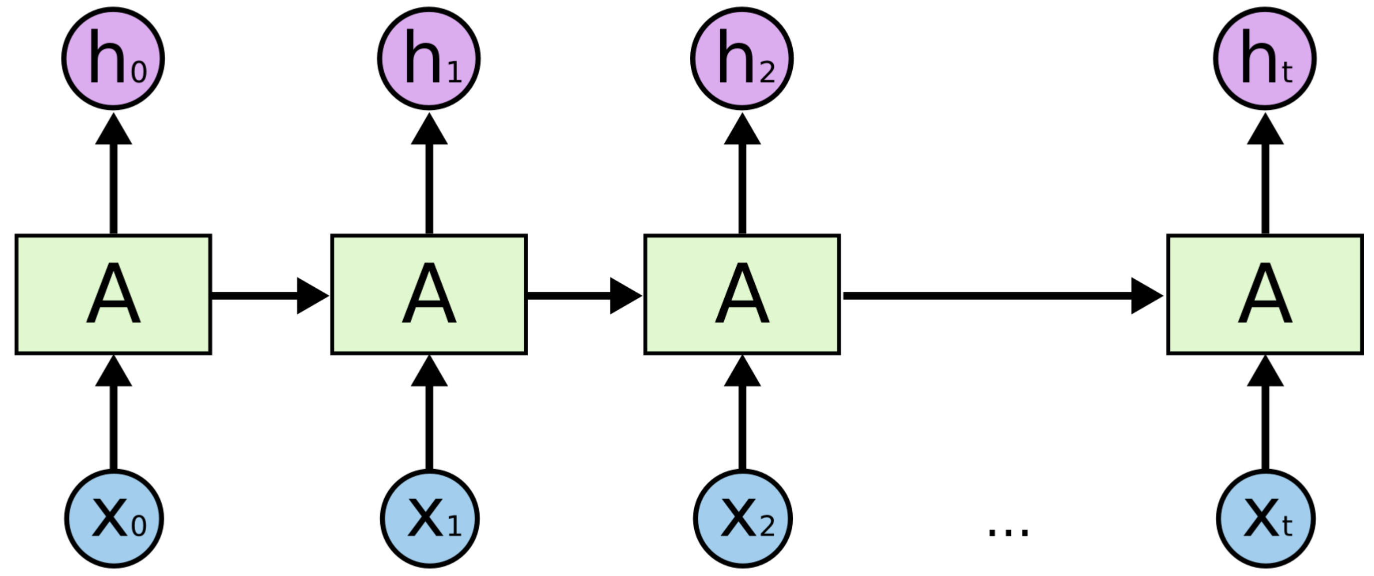




Images taken from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Limitations of Simple Recurrent Unit

- Difficult to train (**gradient vanishing problem**)
- Long distance dependency



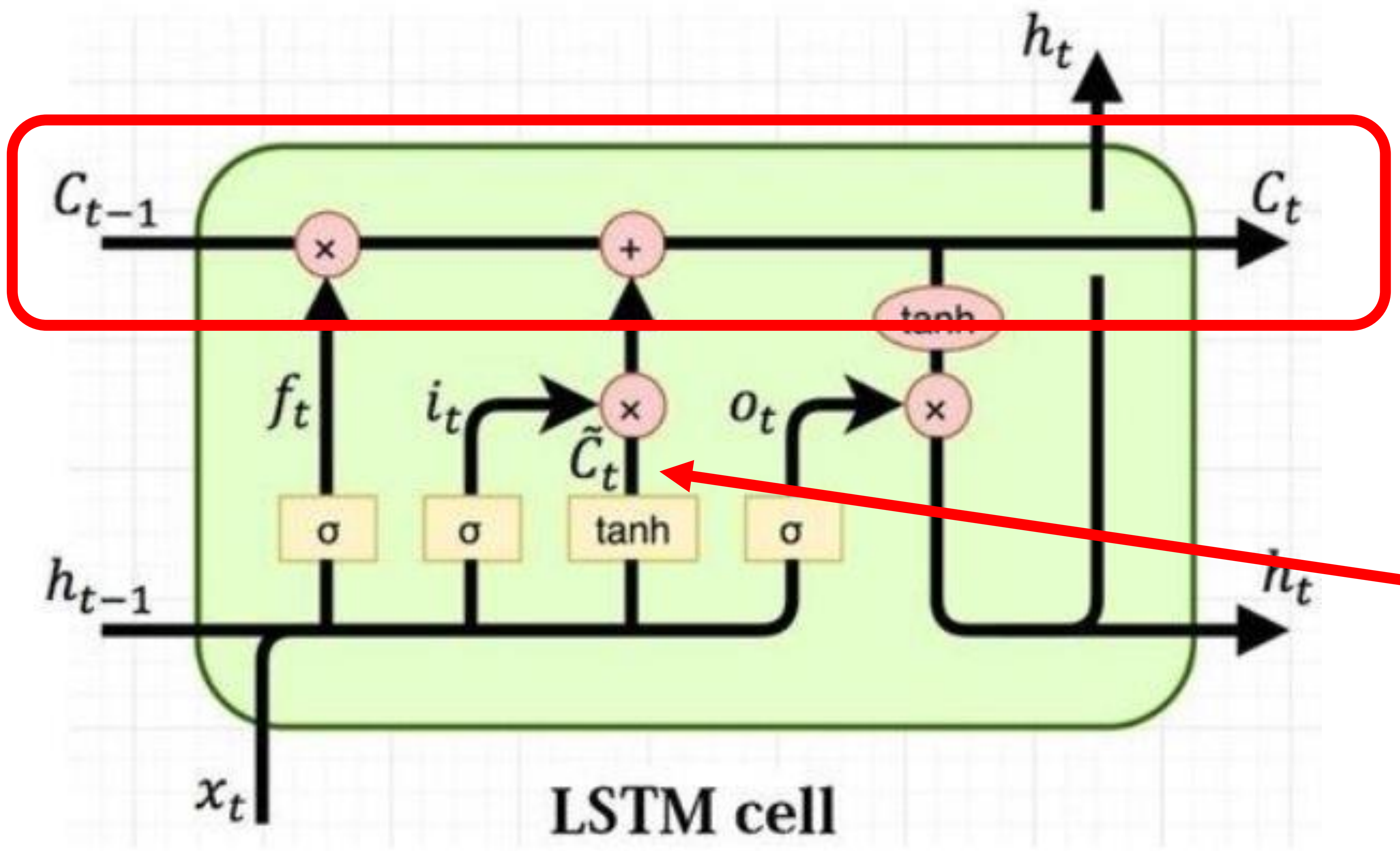
เพิ่ง ลง จาก เครื่อง มา หิว มาก เดี๋ยว ช่วย พา ไป หา อะไร _____ หน่อย

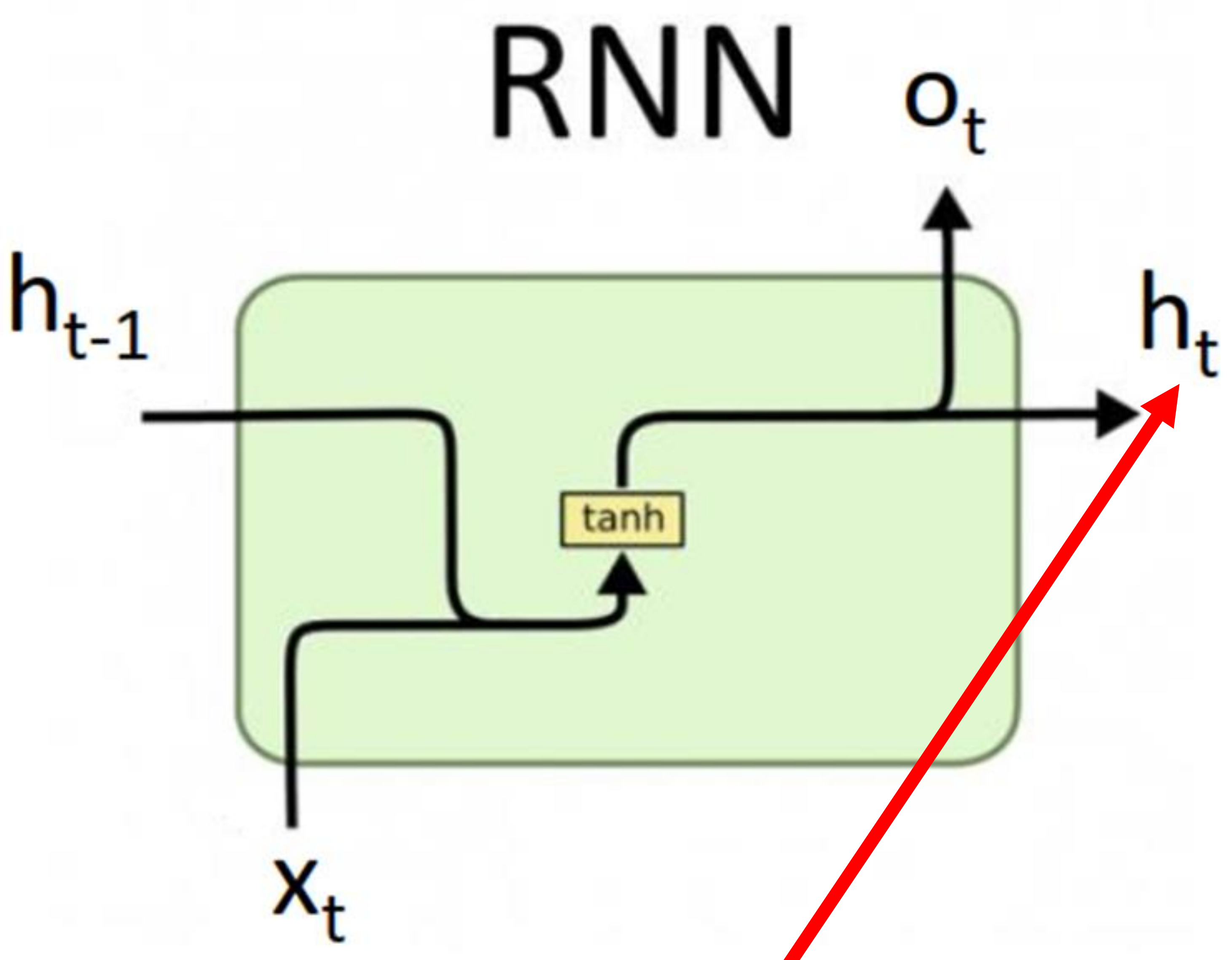
{กิน, ทำ, เล่น, ชม, ...}

Long short-term memory

- Add “***Cell Memory***” **(C_t)** to RNN units
- Cell memory is controlled by **forget gate** and **input gate**.
- Output is controlled by an **output gate**.

Long short-term memory





$$\tilde{C}_t = \tanh(x_t U^g + h_{t-1} W^g)$$

$$C_t = \sigma\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right)$$

$$h_t = \tanh(C_t) * o_t$$

How gates work ?

$$\tilde{C}_t = \tanh(x_t U^g + h_{t-1} W^g)$$

Forget gate

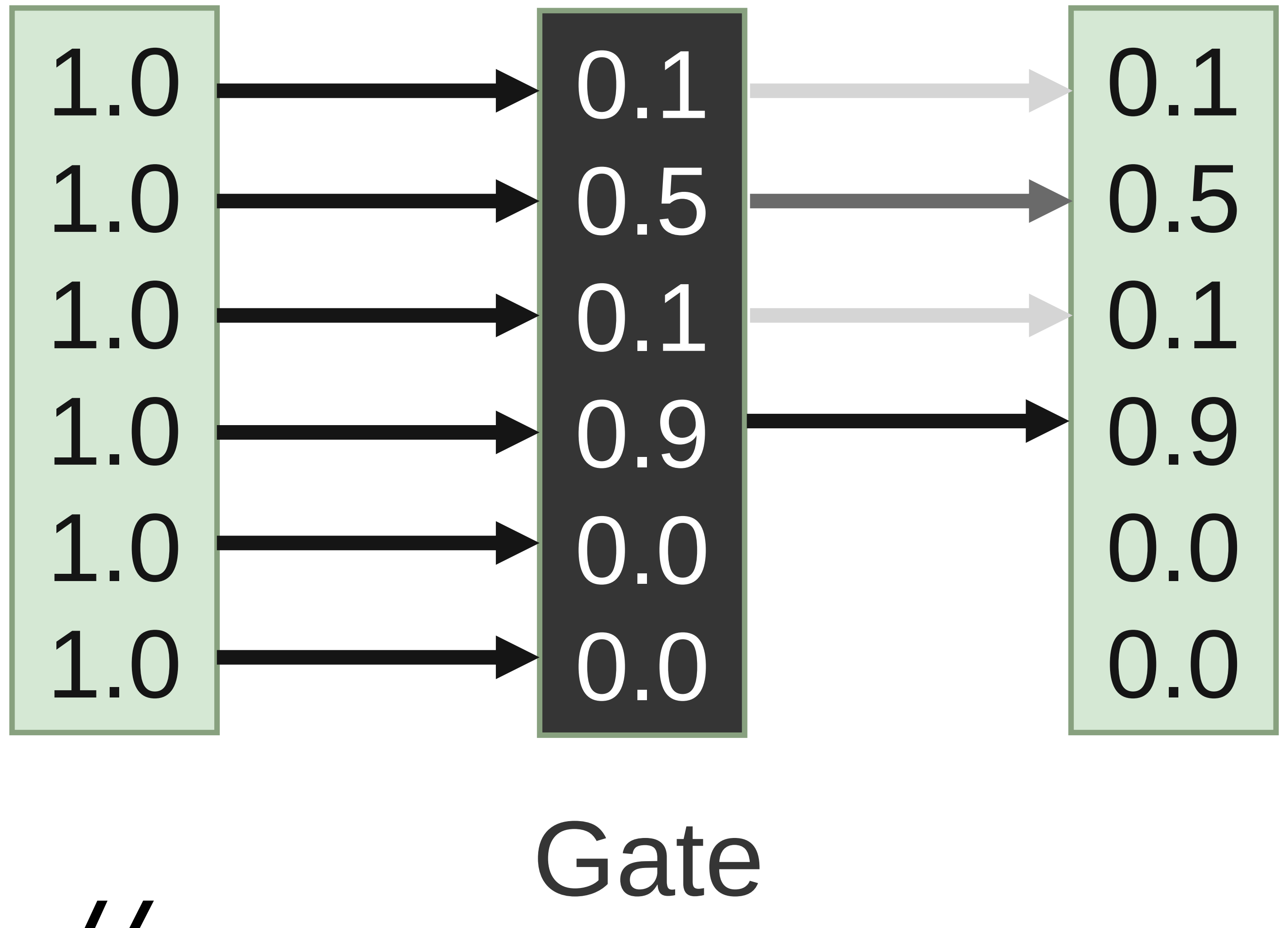
$$C_t = \sigma(\boxed{f_t} * C_{t-1} + \boxed{i_t} * \tilde{C}_t)$$

Input gate

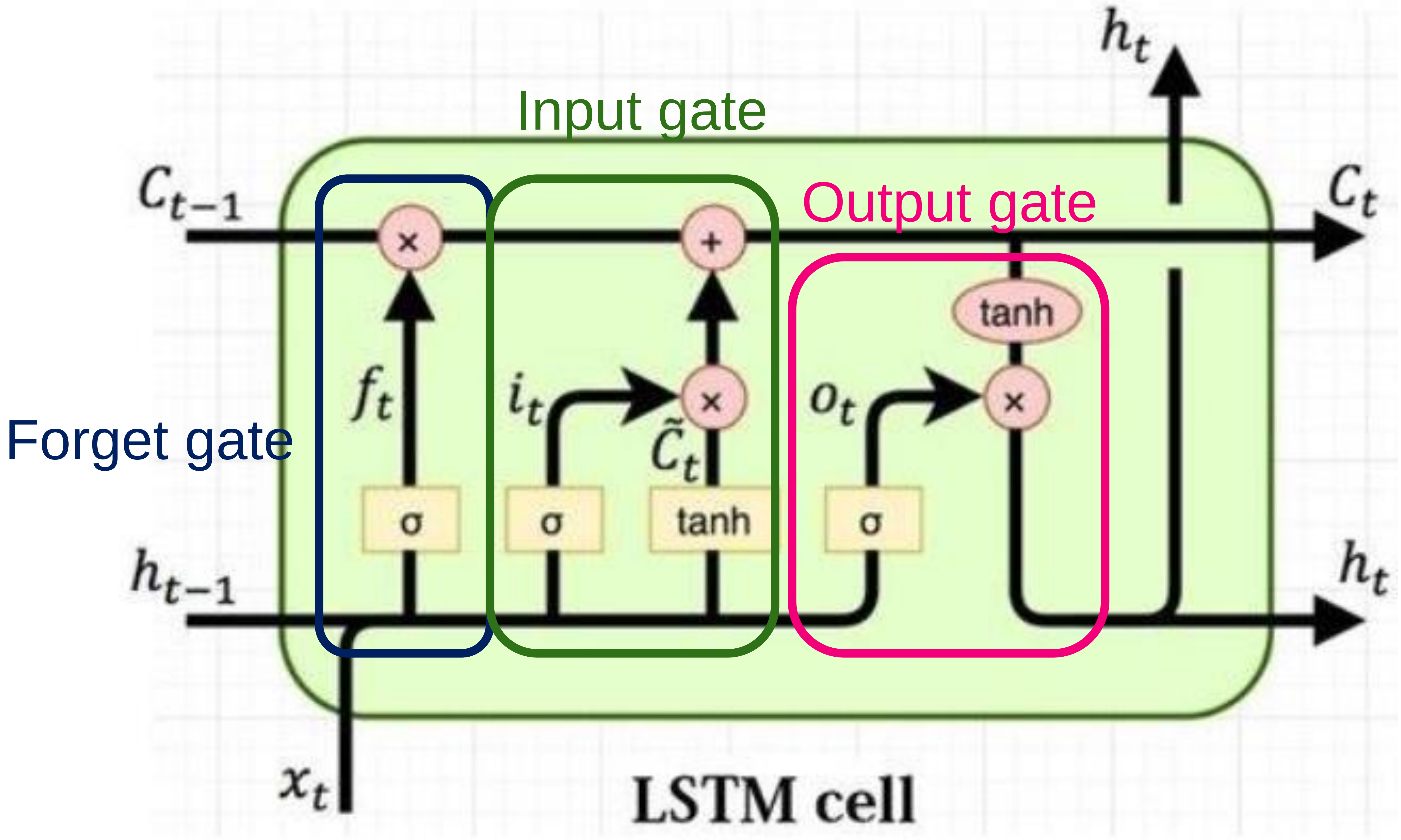
$$h_t = \tanh(C_t) * \boxed{o_t}$$

Output gate

$*$ Element-wise multiplication



How gates work ?

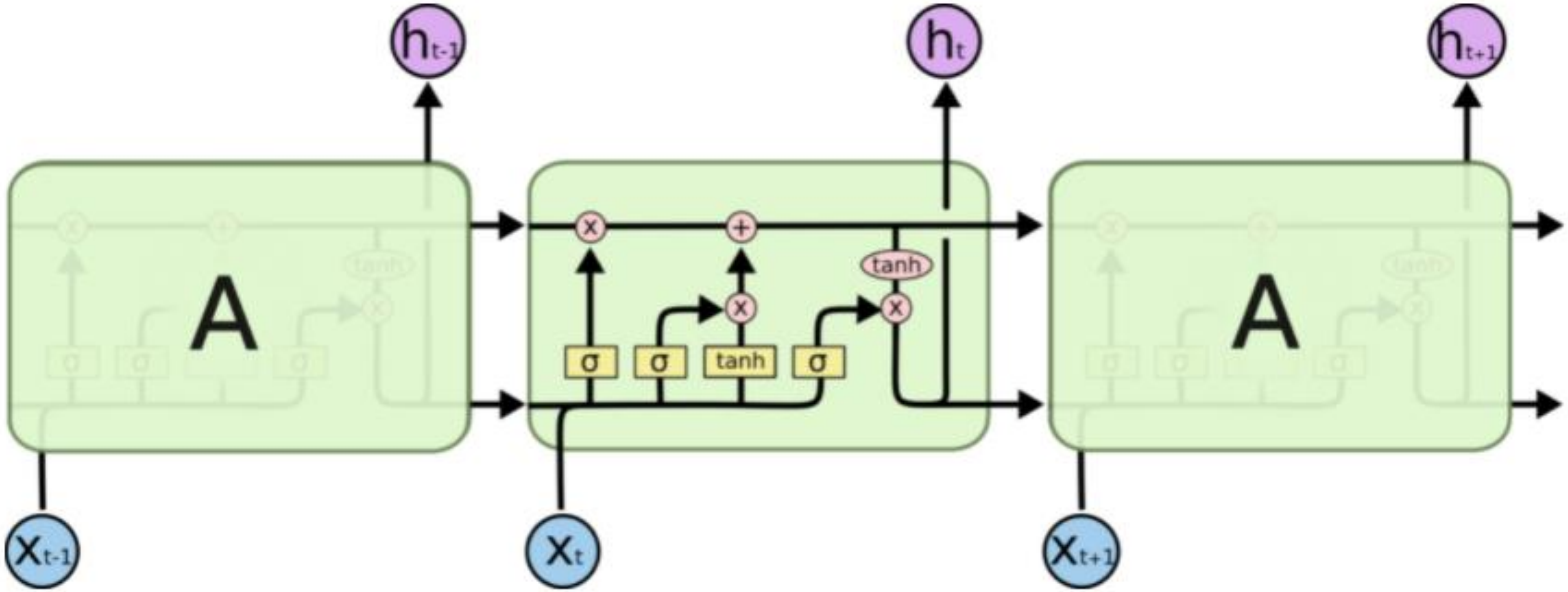


$$i_t = \sigma\left(x_t U^i + h_{t-1} W^i\right)$$
$$f_t = \sigma\left(x_t U^f + h_{t-1} W^f\right)$$
$$o_t = \sigma(x_t U^o + h_{t-1} W^o)$$

σ = sigmoid function

$$\tilde{C}_t = \tanh(x_t U^g + h_{t-1} W^g)$$
$$C_t = \sigma\left(f_t * C_{t-1} + i_t * \tilde{C}_t\right)$$
$$h_t = \tanh(C_t) * o_t$$

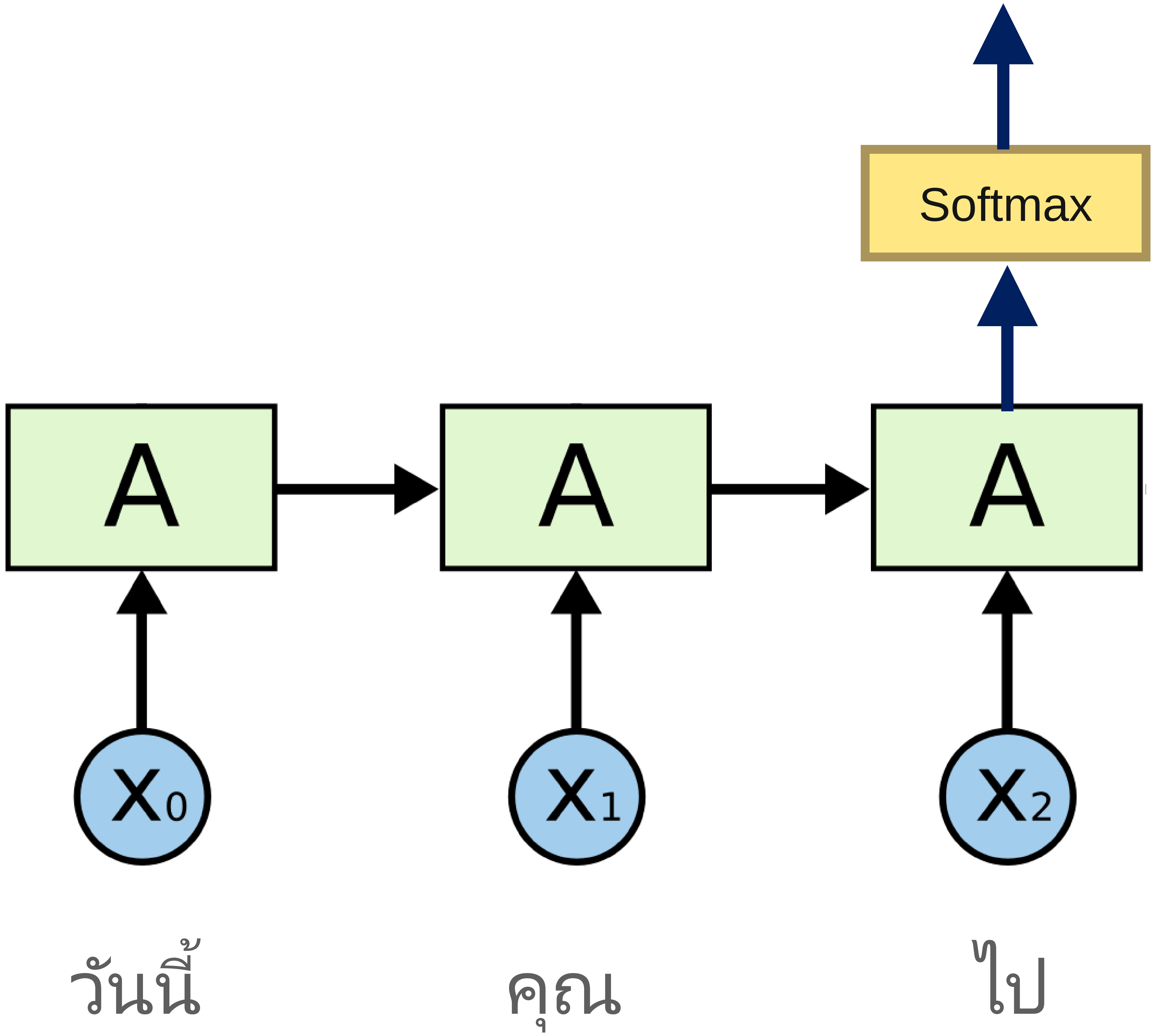
LSTM



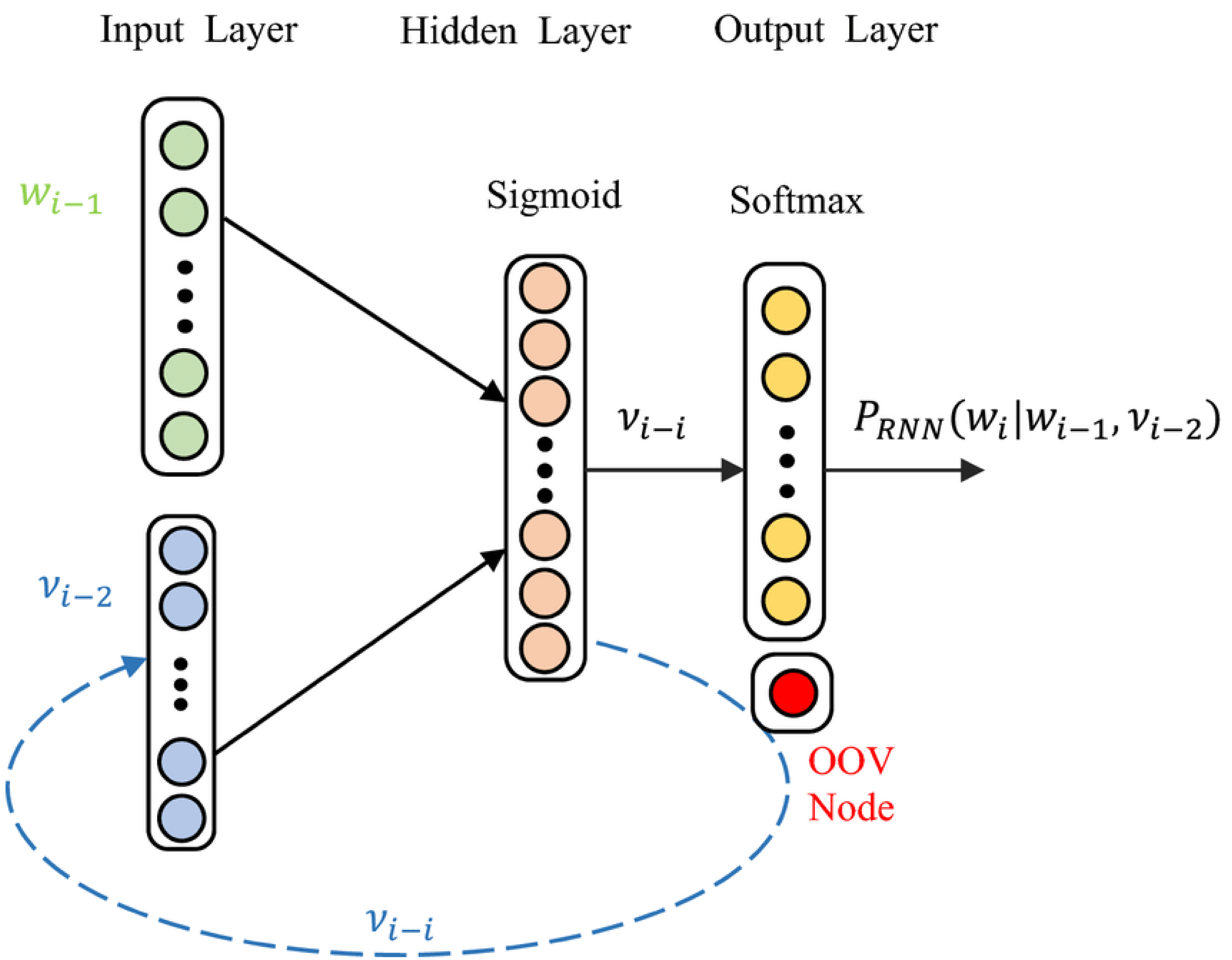
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

RNN-based Language Model

$$p(x_t | x_{t-1}, x_{t-2}, x_{t-3}) = \text{softmax}\,(W h_t)$$



RNN-based Language Model



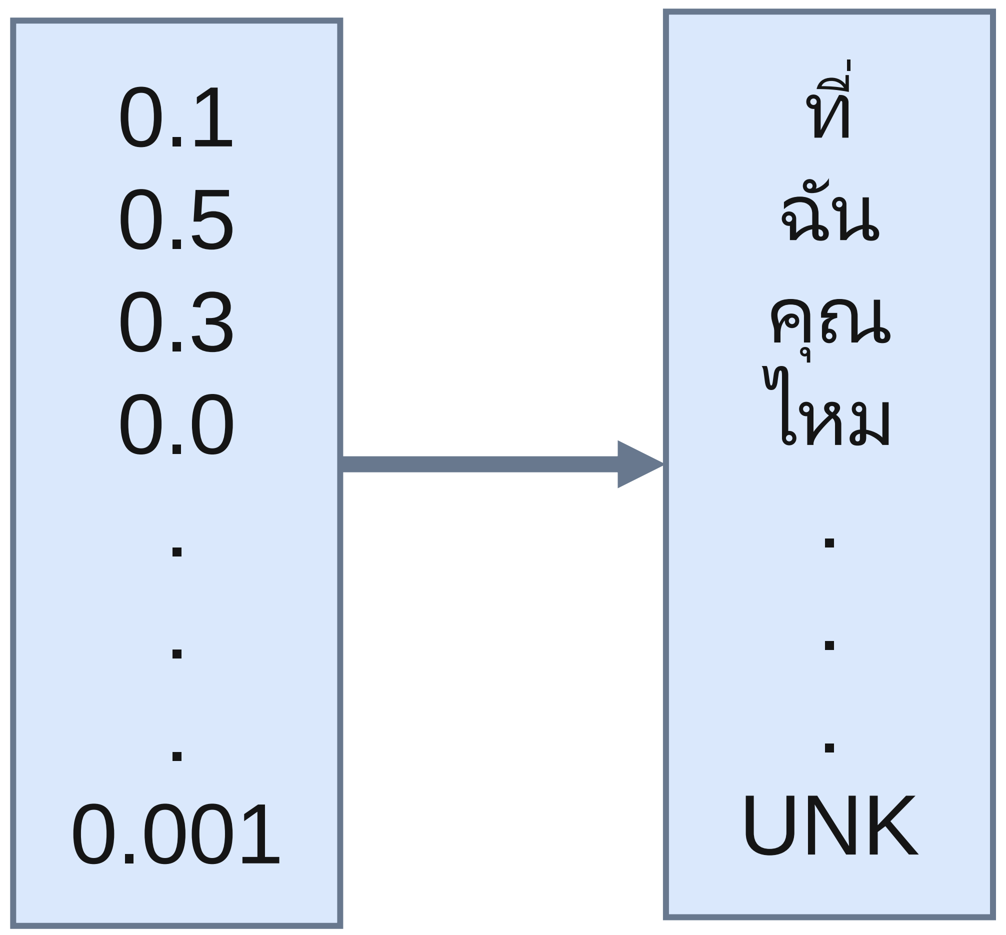


Image taken from https://www.researchgate.net/figure/An-architecture-of-recurrent-neural-network-language-model-for-speech-recognition_fig2_333355077

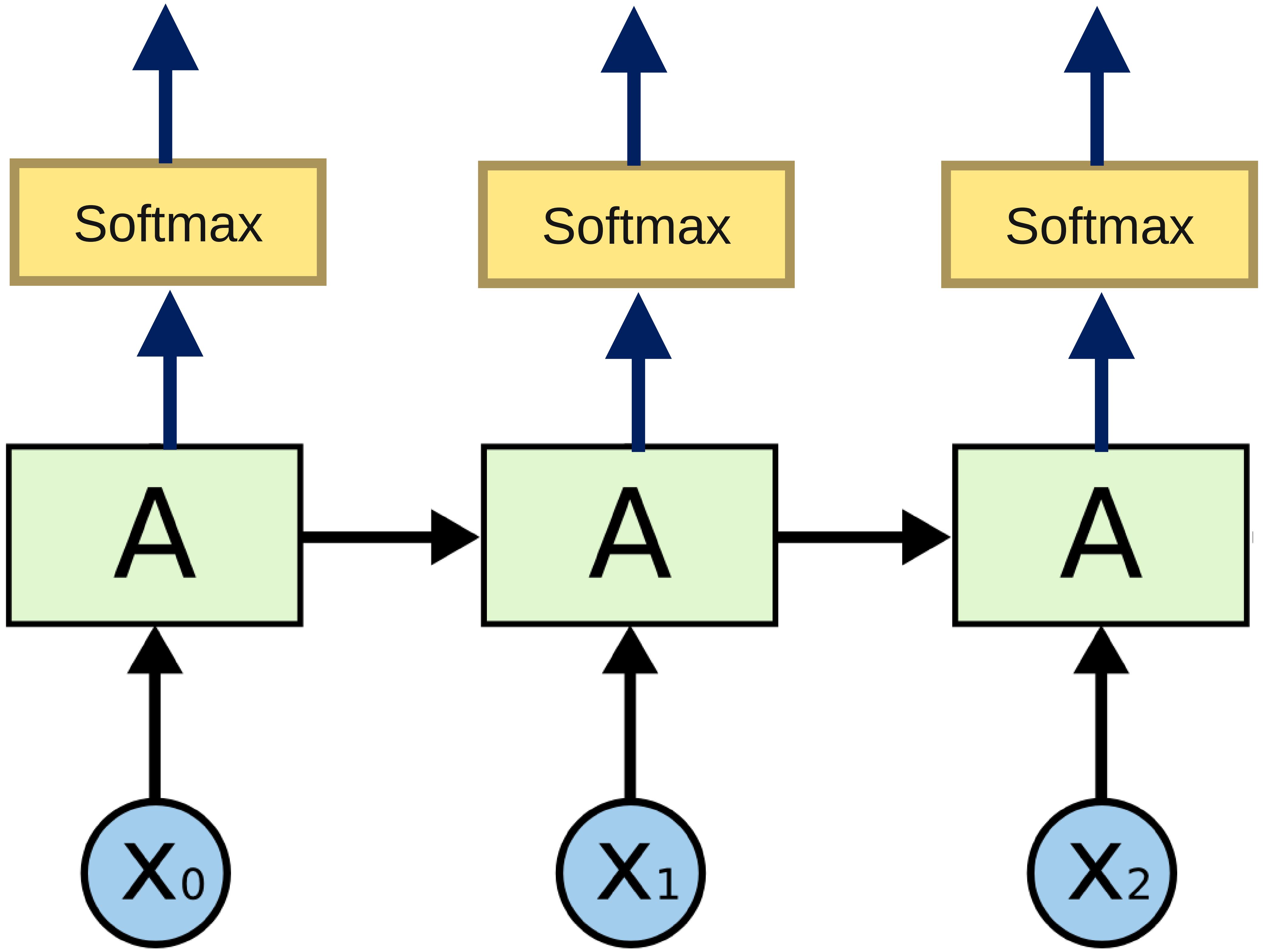
Training RNN-based Language Model

$\Theta = \text{argmax}_{\Theta}\ p(\text{คุณ}|h_0; \Theta) \times p(\text{ไป}|h_1; \Theta) \times p(\text{ไหน}|h_2; \Theta)$

Maximum Likelihood Estimation

$$L(\theta) = \prod_{i=1}^{n} f(x_i|\theta)$$

Θ



วันนี้ คุณ ไป

Negative Log Loss (NLL)

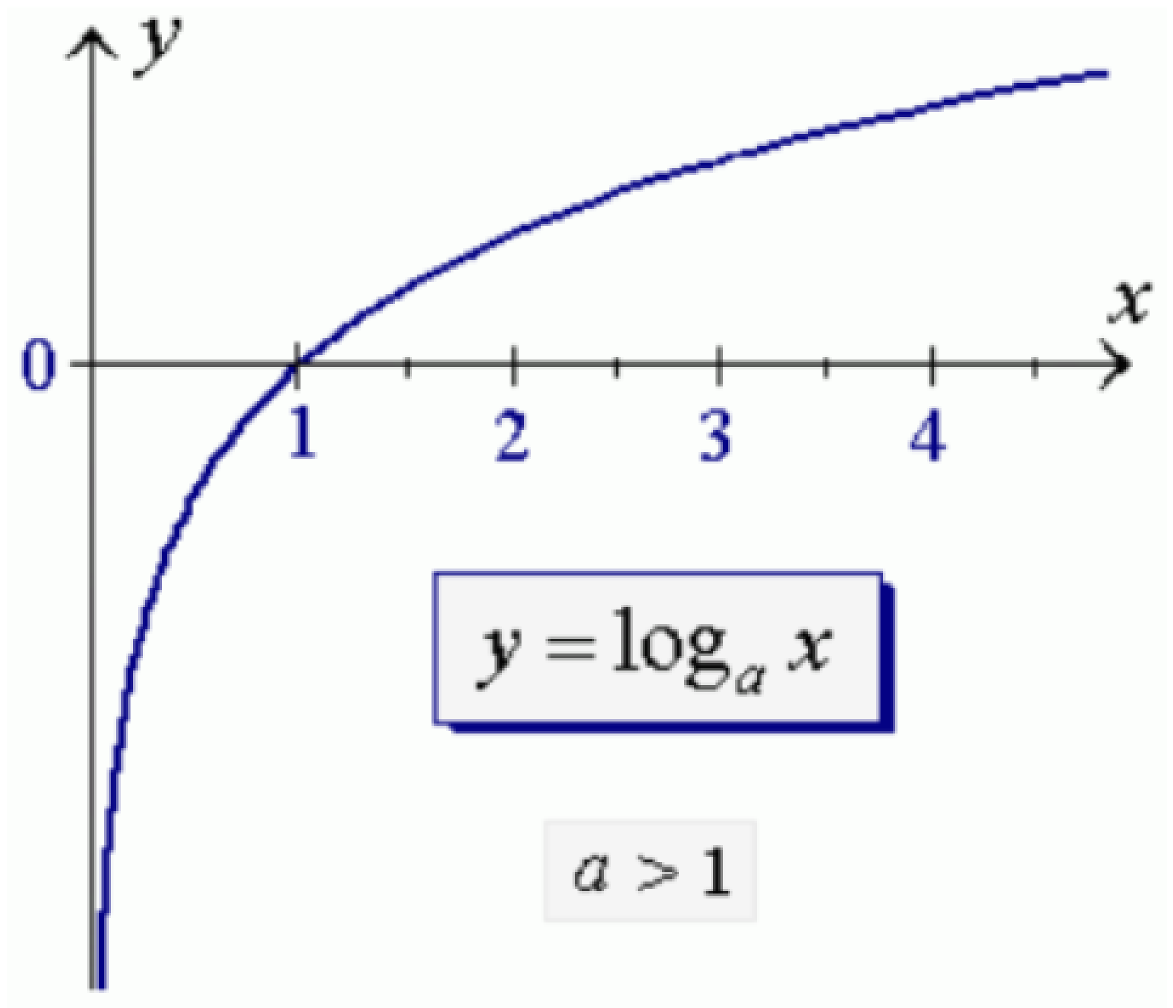
- In many deep learning frameworks, we usually minimize loss functions.

Θ = argmax$_\Theta$ p(คุณ|h_0; Θ) x p(ไป|h_1 ; Θ) x (ไหน|h_2 ; Θ)

Θ = argmax$_\Theta$ log p(คุณ|h_0; Θ) + log p(ไป|h_1 ; Θ) + log p(ไหน|h_2 ; Θ)

Θ = argmin$_\Theta$ - log p(คุณ|h_0; Θ) - log p(ไป|h_1 ; Θ) - log p(ไหน|h_2 ; Θ)

$$\mathrm{NLL} = -\sum_{t=1}^{T}\sum_{j=1}^{V} \mathrm{y}_j^t \log \hat{y}_j^t$$



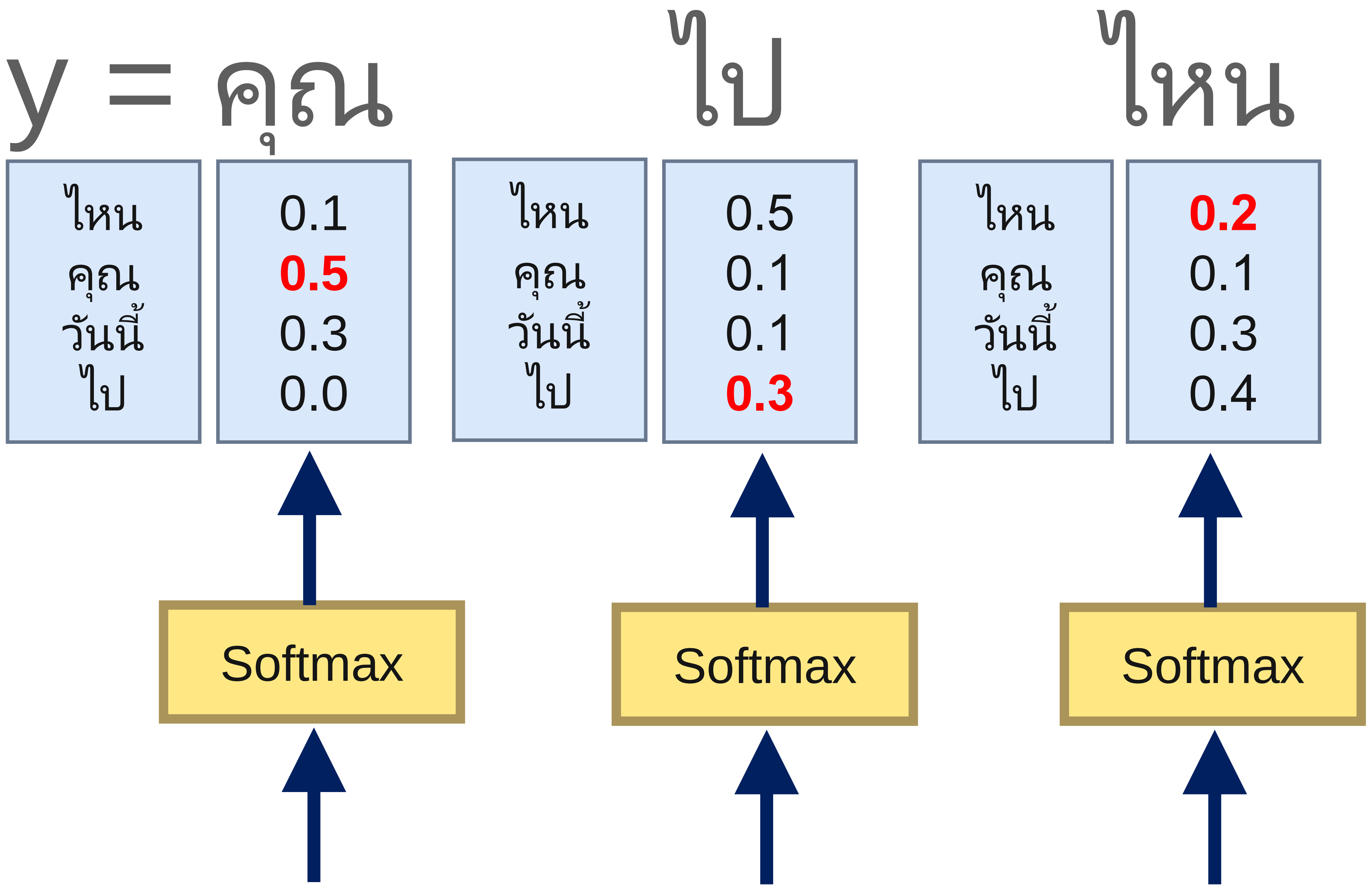
Negative Log Loss (NLL)

$$\mathrm{NLL} = -\sum_{\mathrm{t}=1}^{\mathrm{T}} \sum_{j=1}^{V} \mathrm{y}_j^t \log \hat{y}_j^t$$

- Because y^t are one-hot vectors [0 , 0 , …, 1, …, 0, 0] , only the position of the ground truth is 1, therefore

$$\mathrm{NLL} = -\sum_{t=1}^{T} \log p(y^t | h_t; \theta)$$

Negative Log Loss (NLL)



$$\text{NLL} = -\sum_{t=1}^{T} \log p(y^t | h_t; \theta)$$

$$= -(\log 0.5 + \log 0.3 + \log 0.2)$$

Sequence-to-Sequence

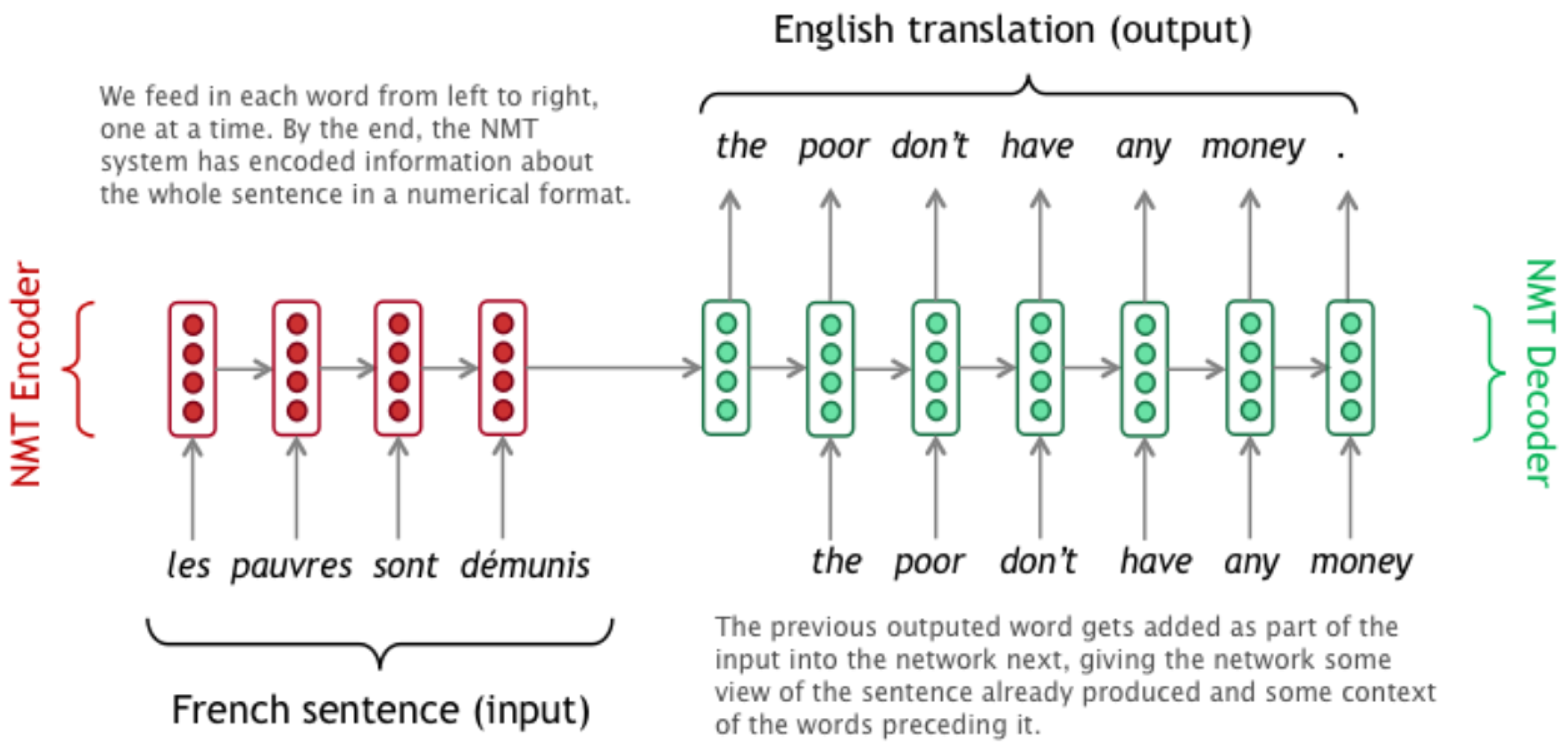
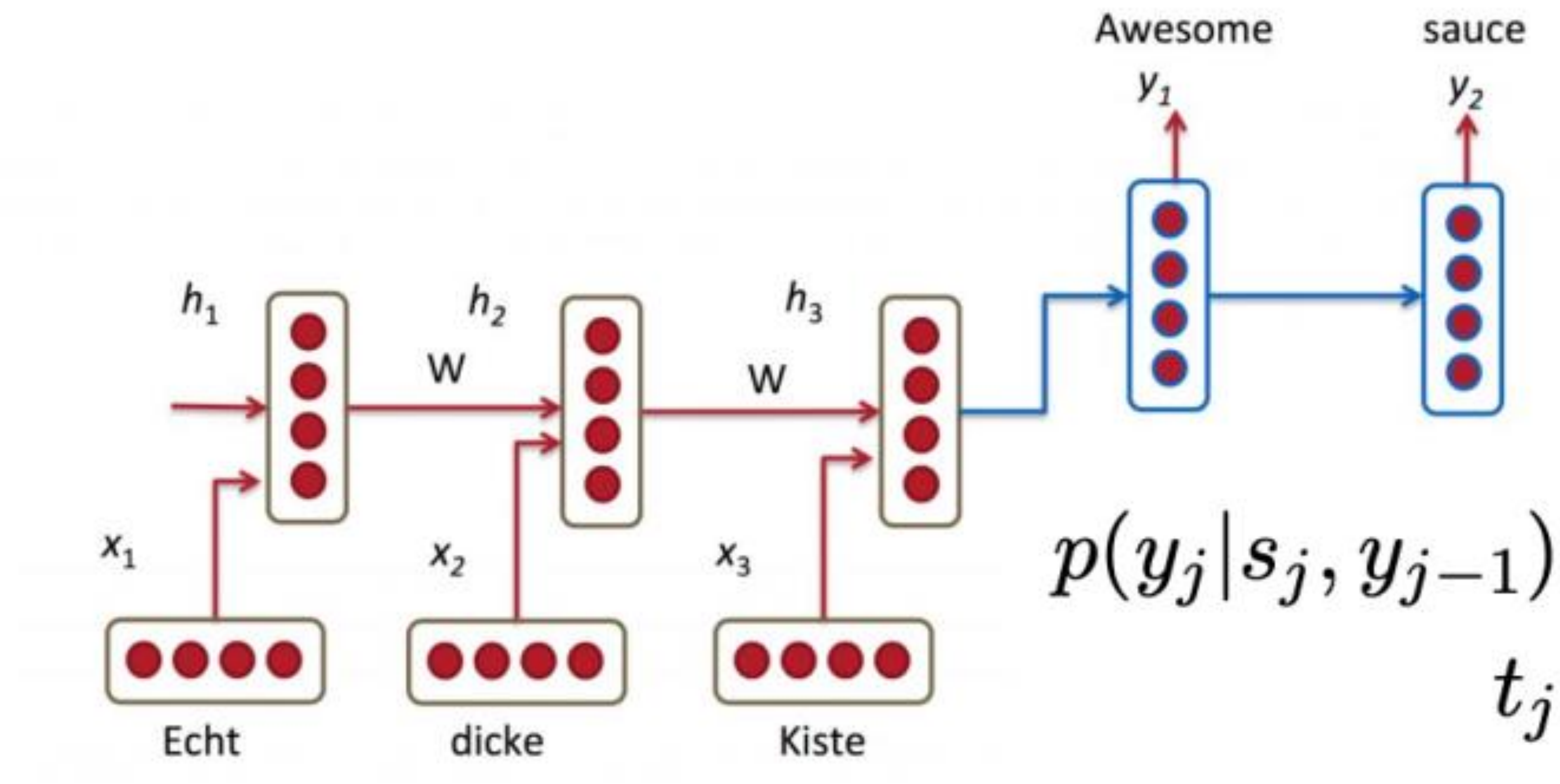


Image taken from https://medium.com/analytics-vidhya/seq2seq-models-french-to-english-translation-using-encoder-decoder-model-with-attention-9c05b2c09af8

Sequence-to-Sequence

- Encoder : reads source input tokens one-by-one and produce a context vector which represents the source sentence.
- Decoder : generates target tokens from left to right using the context vector and previous output (similar to what language modeling does)



$$p(y_j|s_j, y_{j-1}) = \mathrm{softmax}(W_o t_j)$$
$$t_j = \tanh(W_{t1} s_j + W_{t2} \mathrm{E}[y_{j-1}])$$

Image taken from : http://cs224d.stanford.edu/lectures/CS224d-Lecture8.pdf

Sequence-to-Sequence

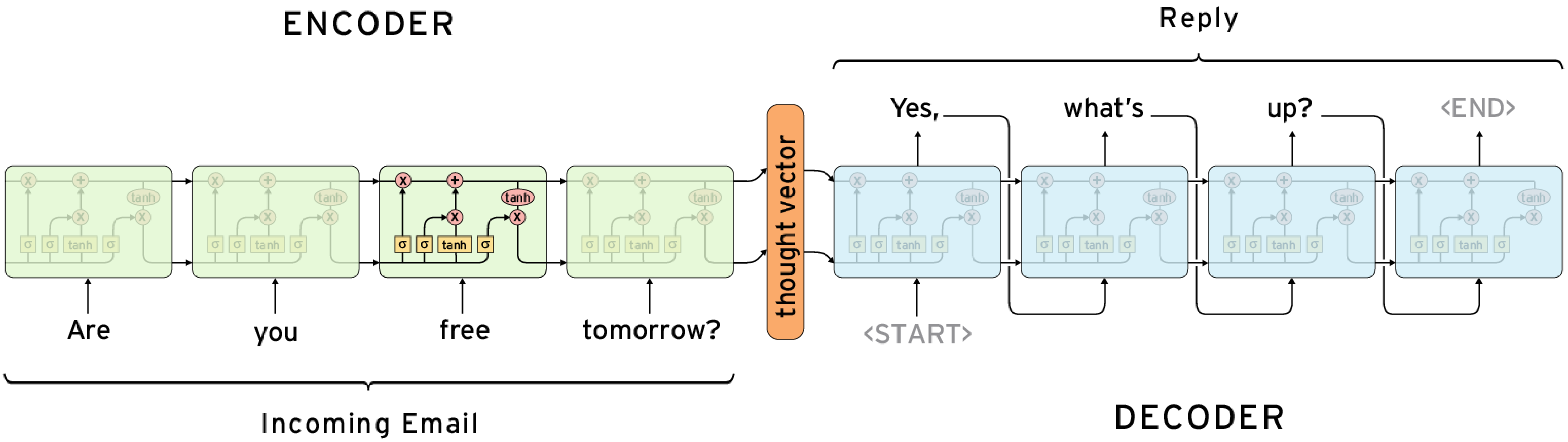
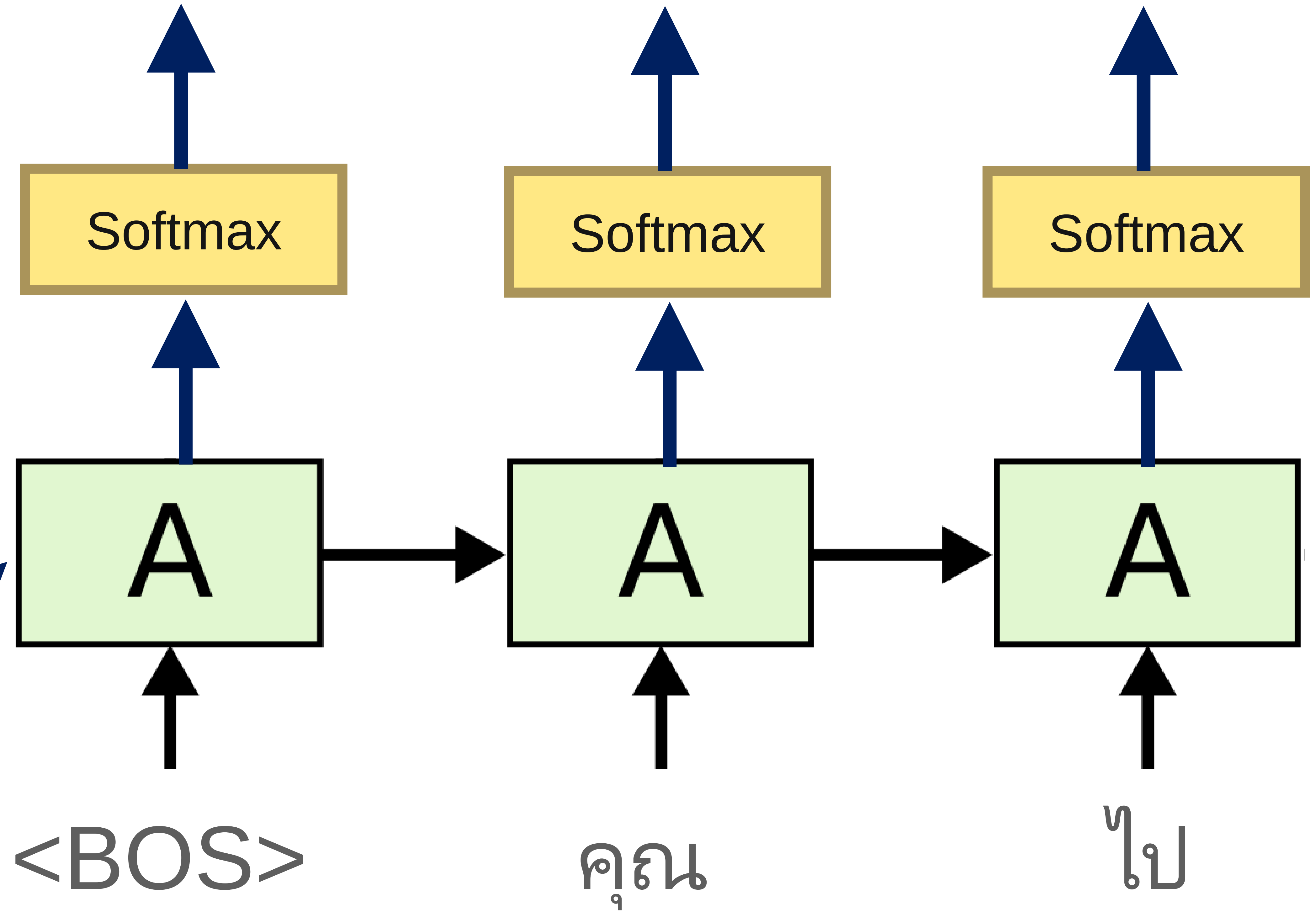


Image taken from suriyadeepan.github.io

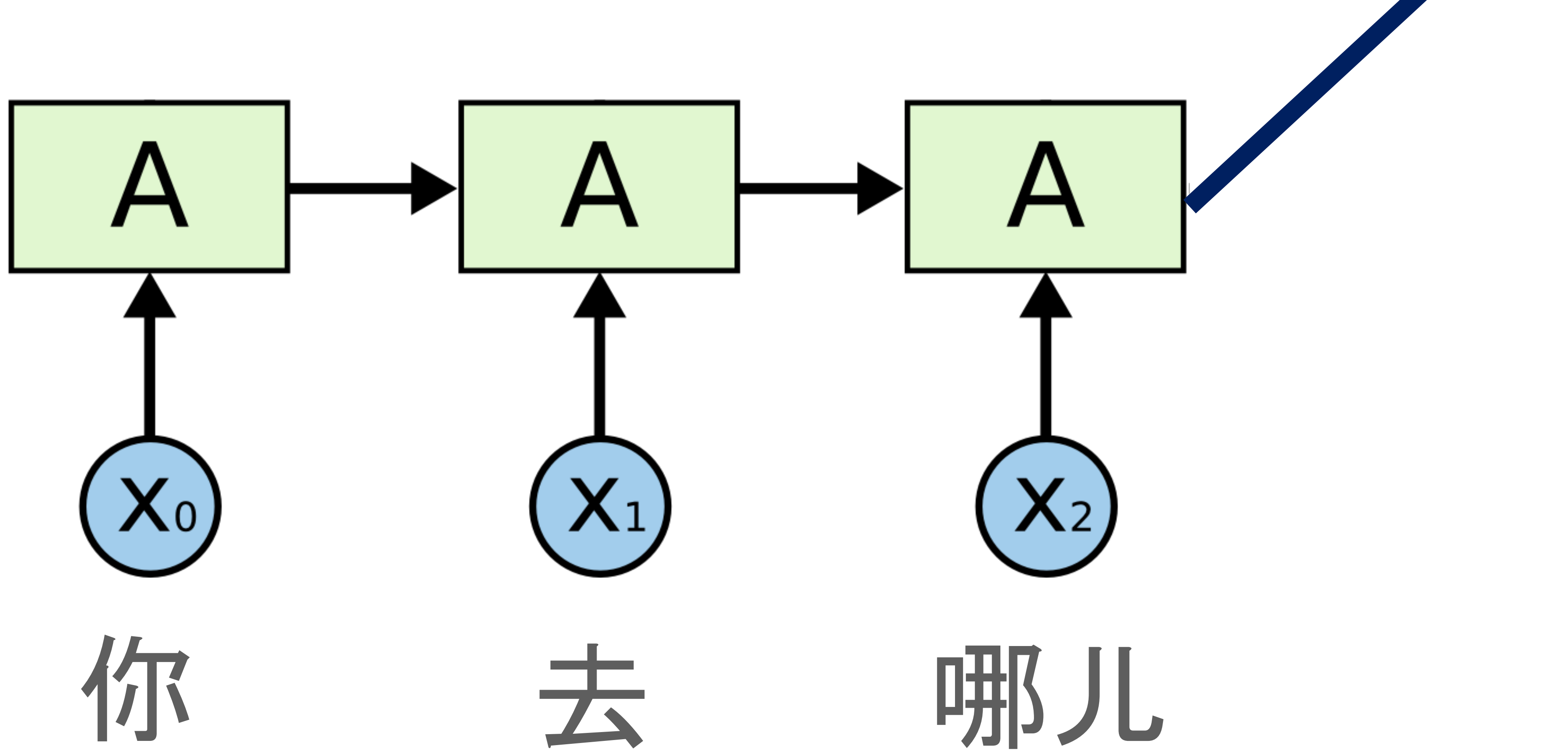
Training Sequence-to-Sequence

$\Theta = \text{argmax}_{\Theta}\ p(\text{คุณ}|h_0; \Theta) \times p(\text{ไป}|h_1 ; \Theta) \times p(\text{ไหน}|h_2 ; \Theta)$



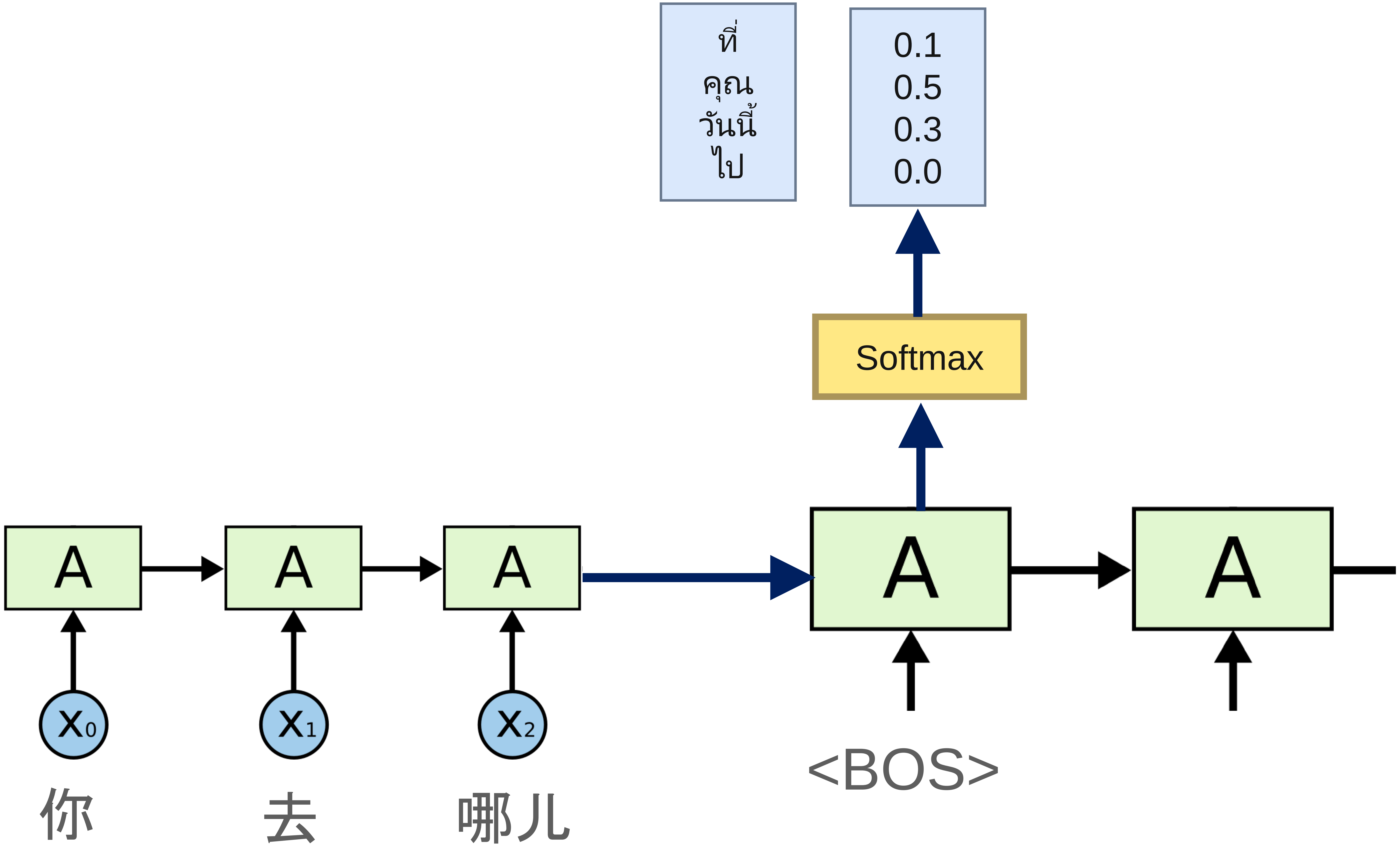
Maximum Likelihood Estimation

$$L(\theta) = \prod_{i=1}^{n} f(x_i|\theta)$$

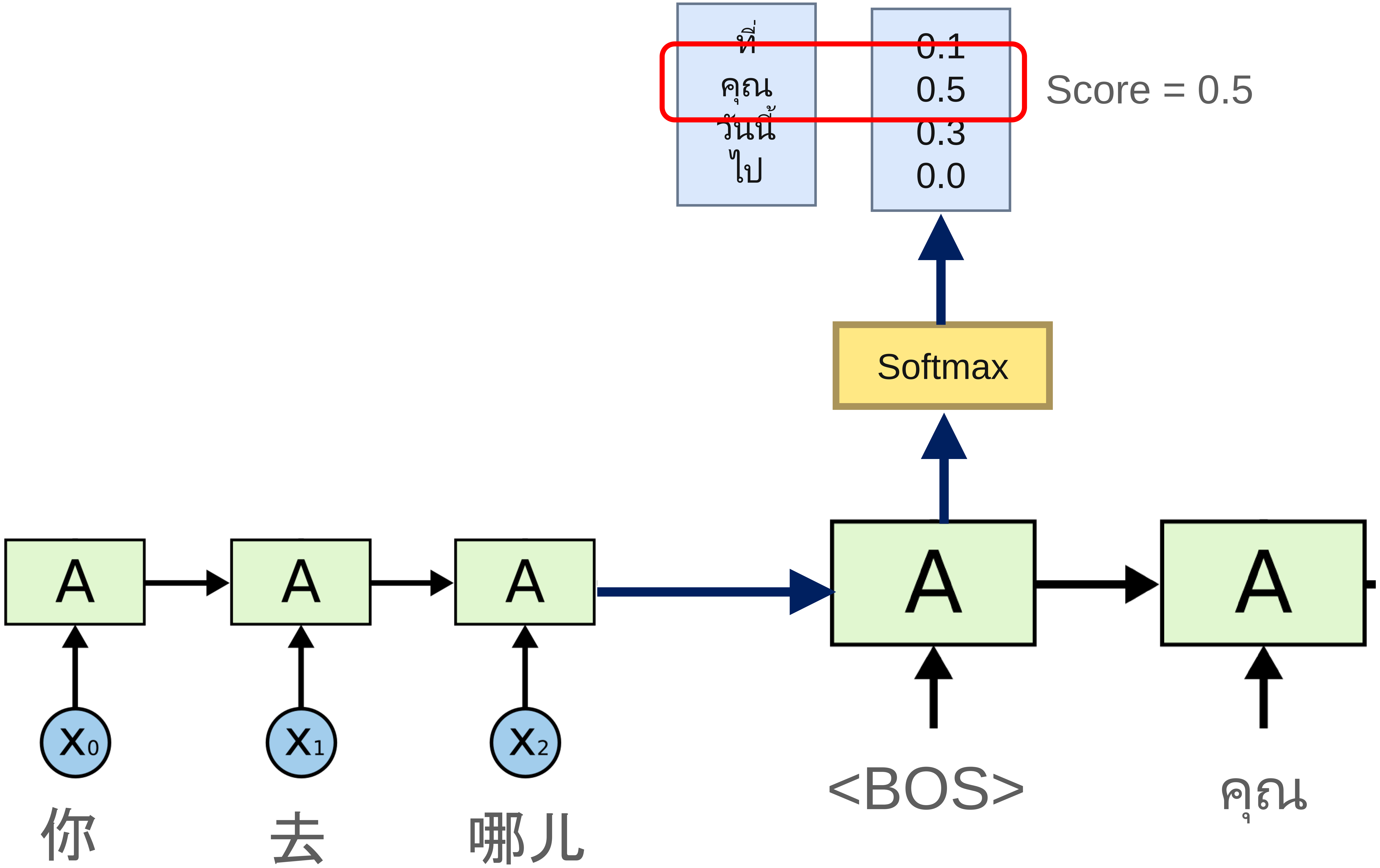


$$\Theta = \{\Theta_E, \Theta_D\}$$

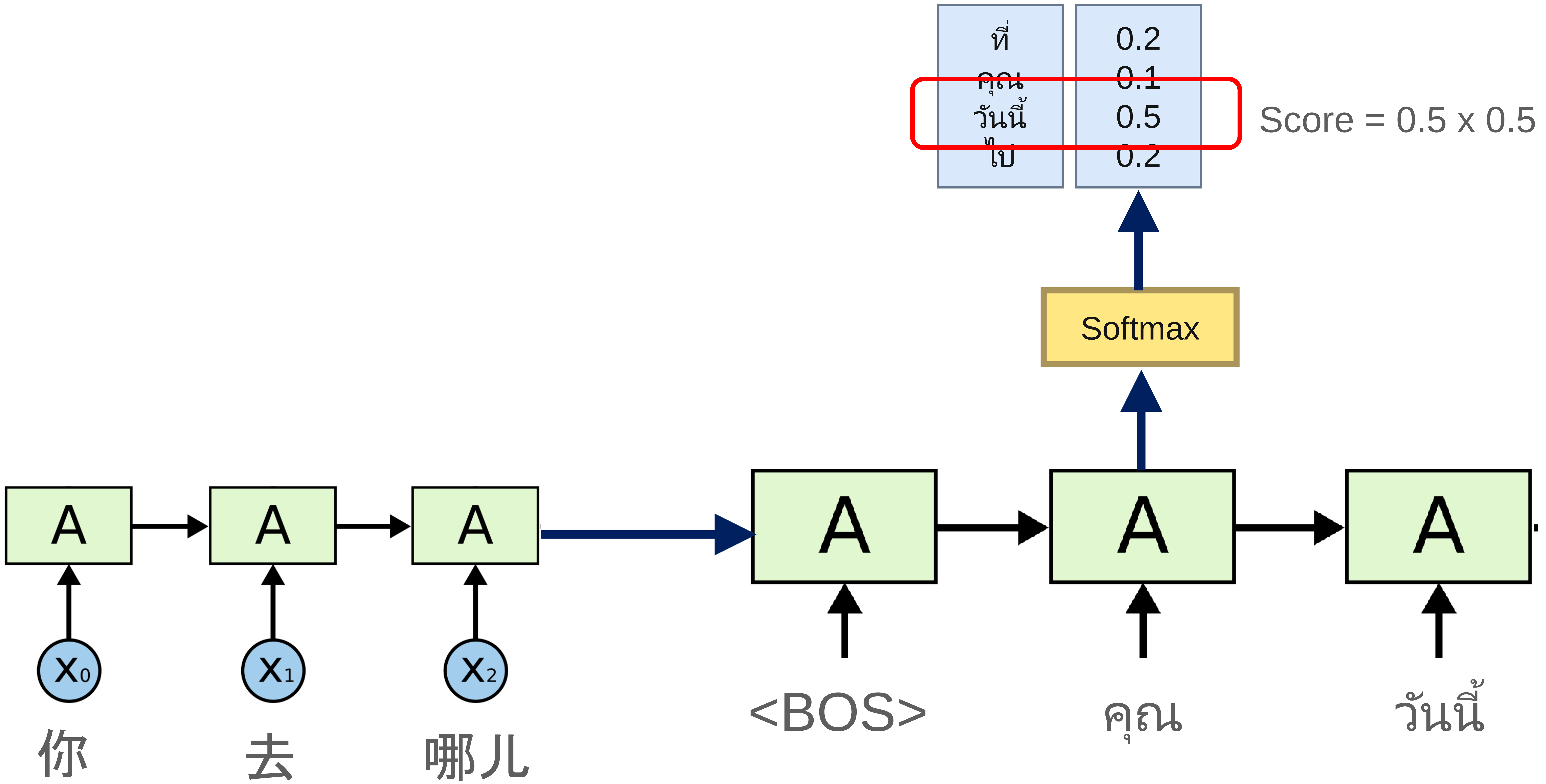
Decoding – Greedy



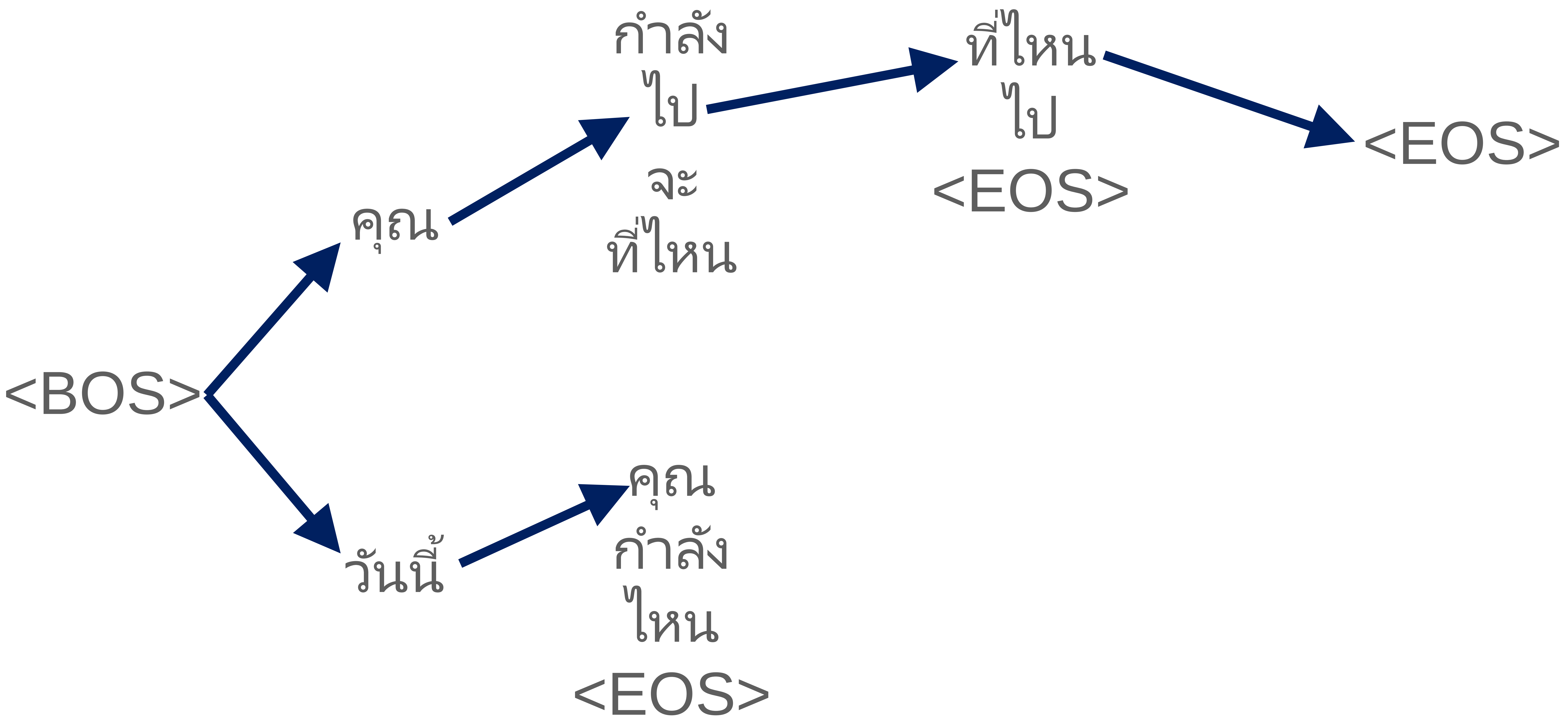
Decoding – Greedy



Decoding – Greedy



Decoding – Beam Search



Summary

- Recurrent Neural Network
- RNN-based Language Model
- Seq2Seq

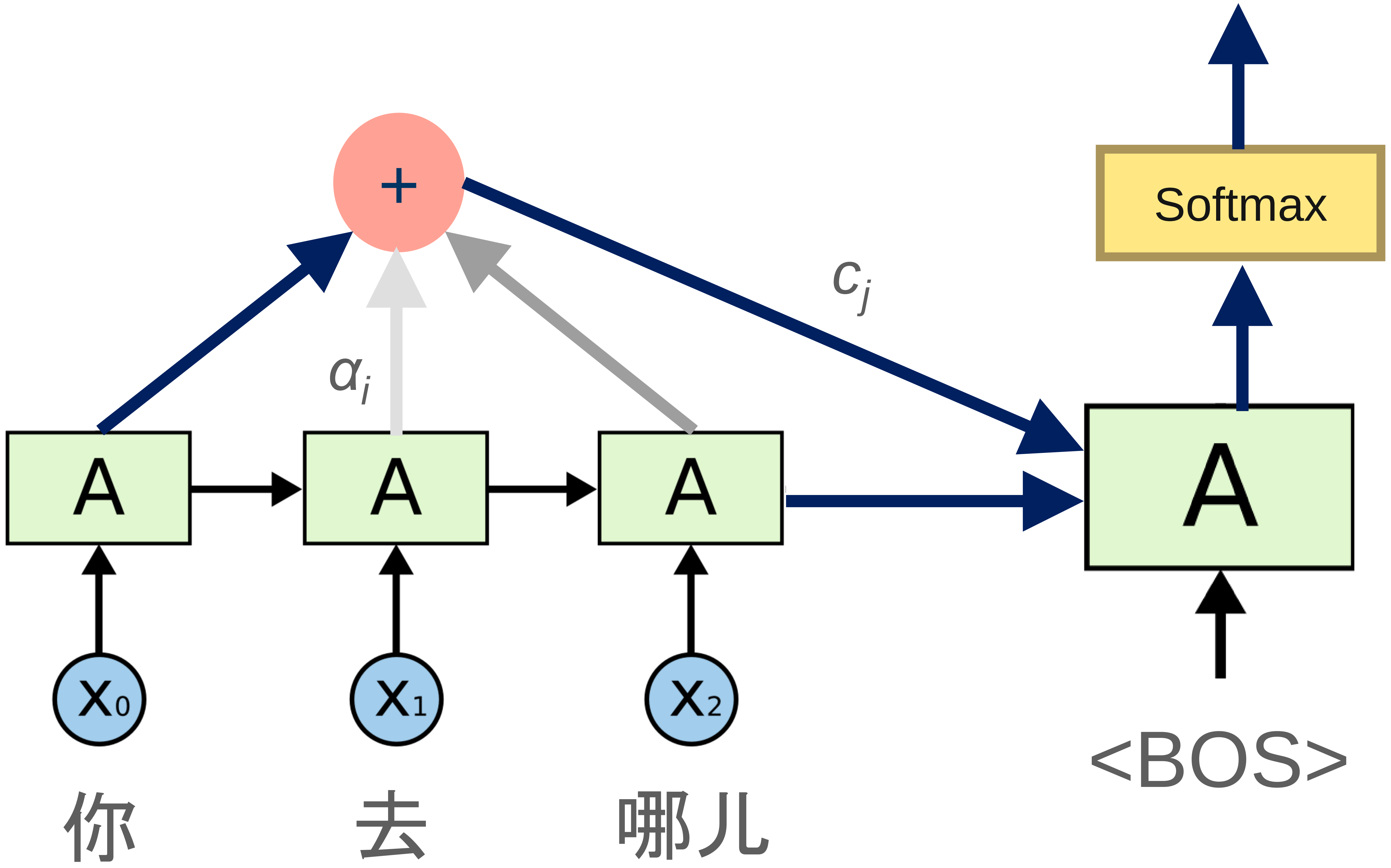
Neural Machine Translation with Attention

Limitations of Seq2Seq Model

- Inefficient for long sentences
- Only good for language pairs with less grammatical variations

Attention

$$p(y_j \mid s_j, y_{j-1}, c_j) = \mathrm{softmax}(W_o t_j)$$
$$t_j = \tanh(W_{t1} s_j + W_{t2} \mathrm{E}[y_{j-1}] + W_{t3} c_j)$$

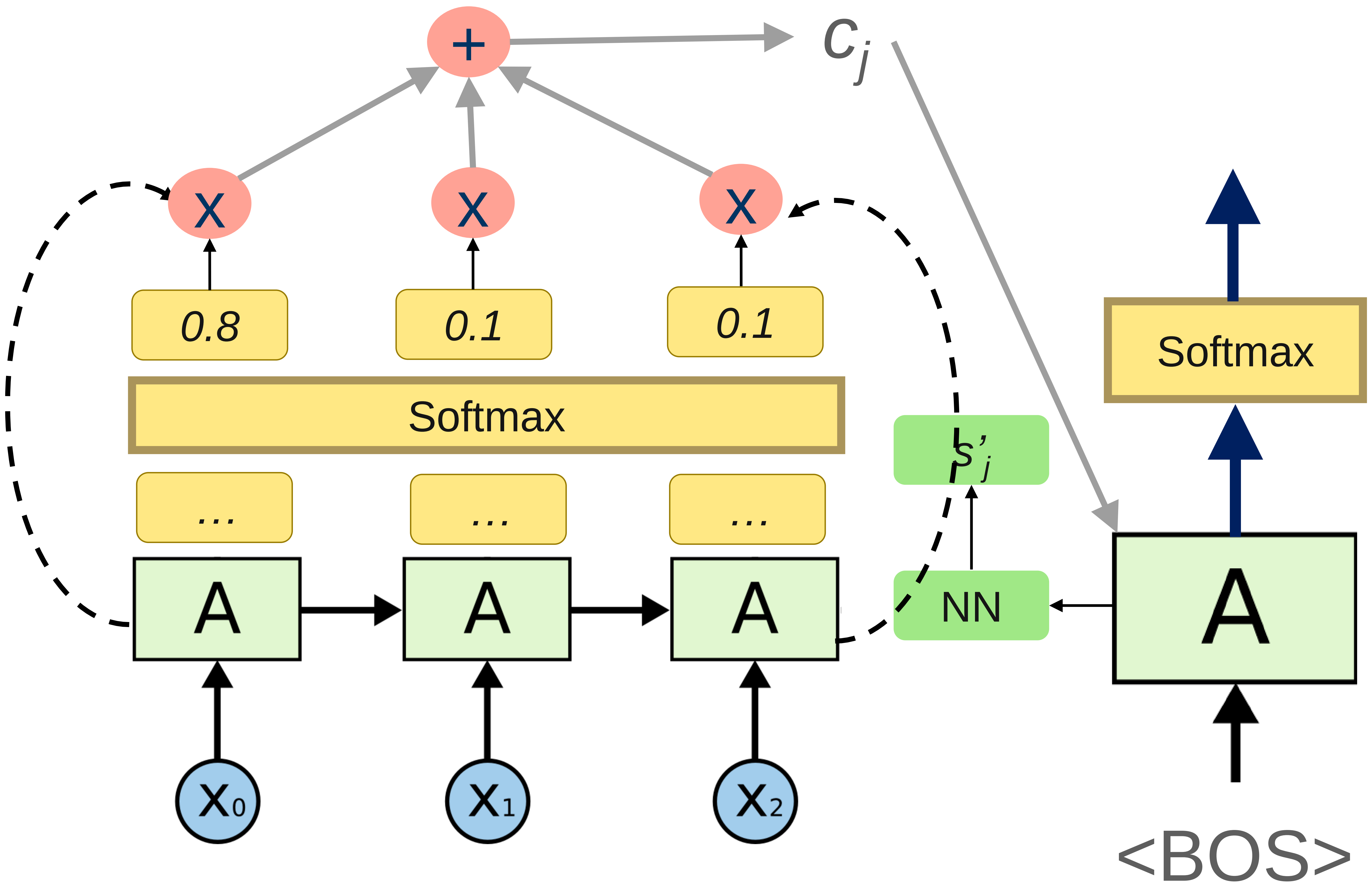


Attention

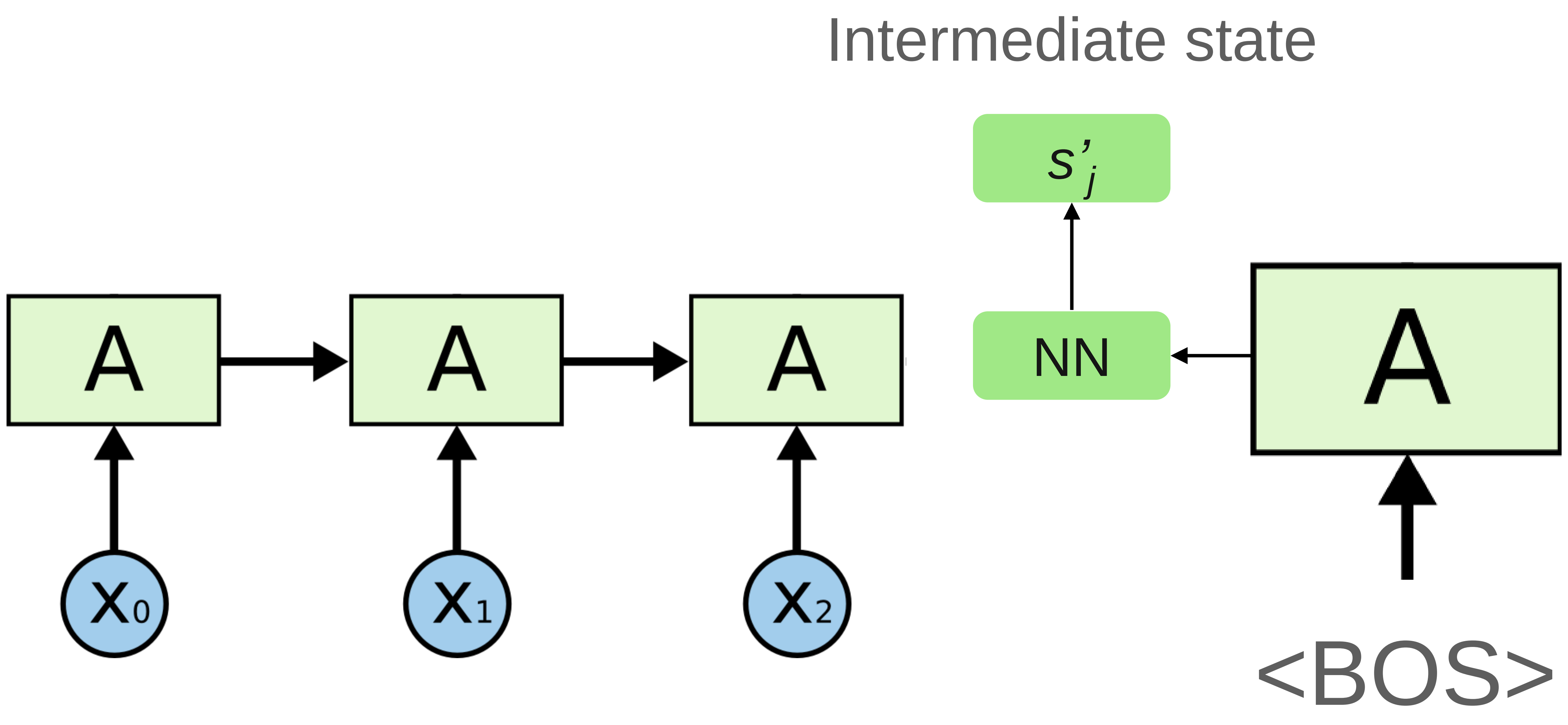
$$c_j = \mathrm{ATT}(C, s'_j) = \sum_{i=1}^{I} \alpha_{ij} h_i,$$

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

$$e_{ij} = v_a^{\mathrm{T}} \tanh(U_a s'_j + W_a h_i)$$

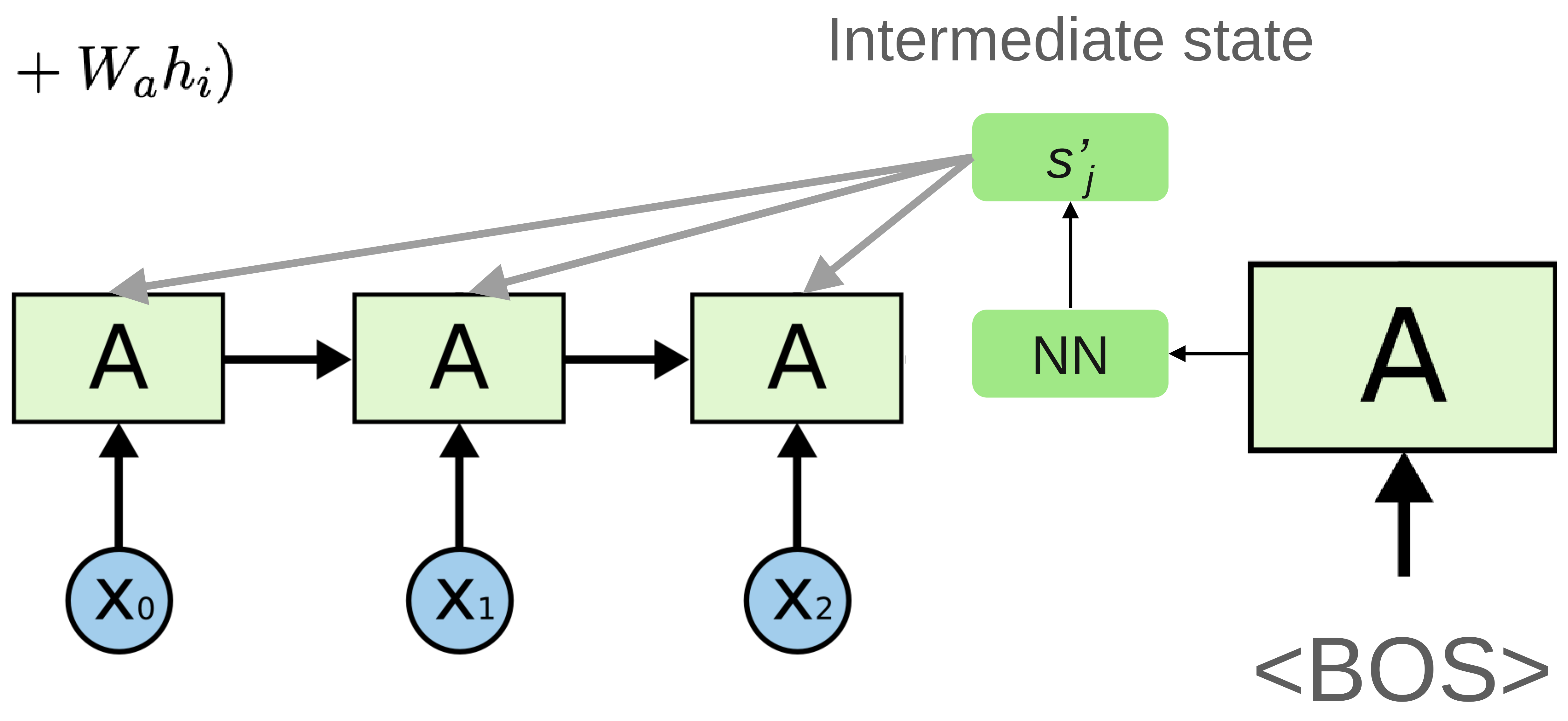


Attention – Step 1



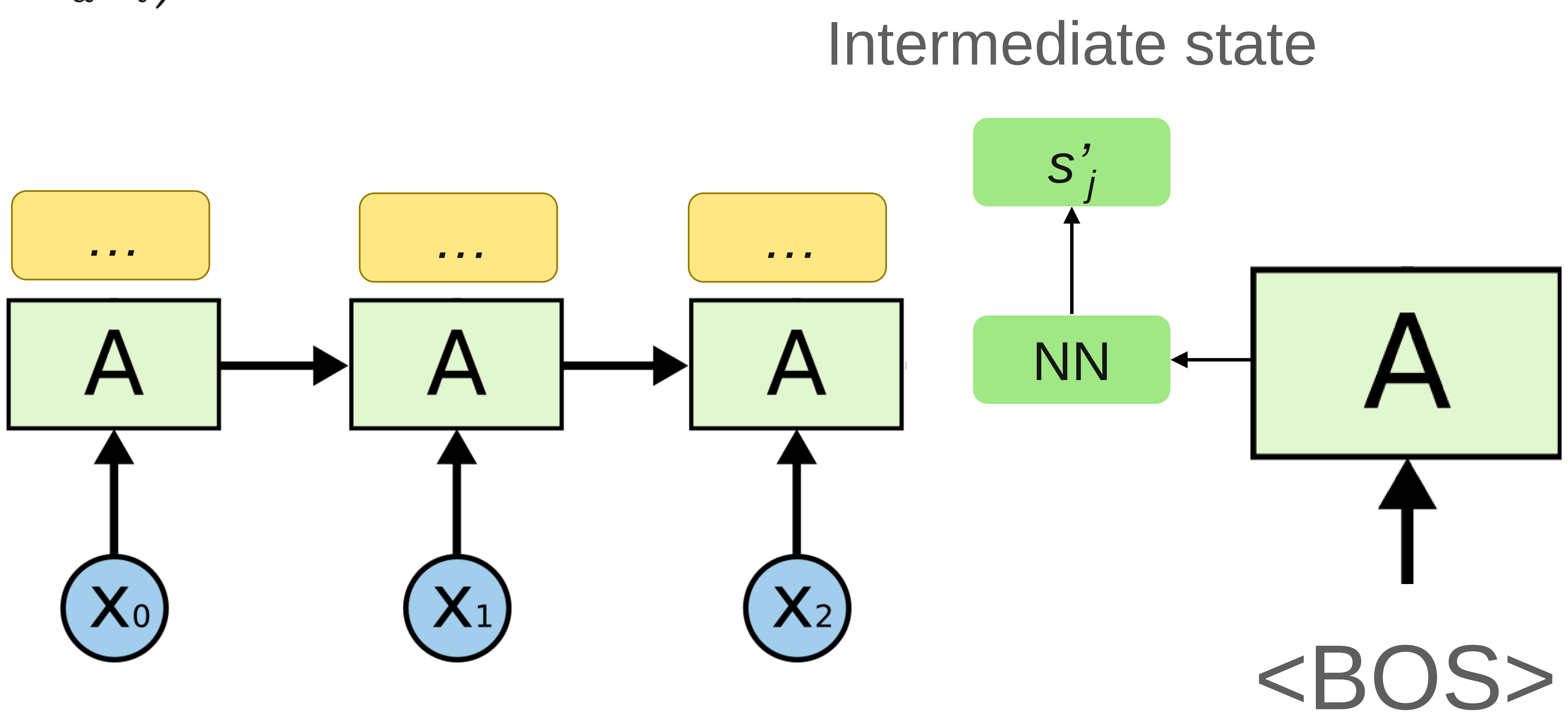
Attention – Step 2

$$e_{ij} = v_a^{\mathrm{T}} \tanh(U_a s'_j + W_a h_i)$$



Attention – Step 2

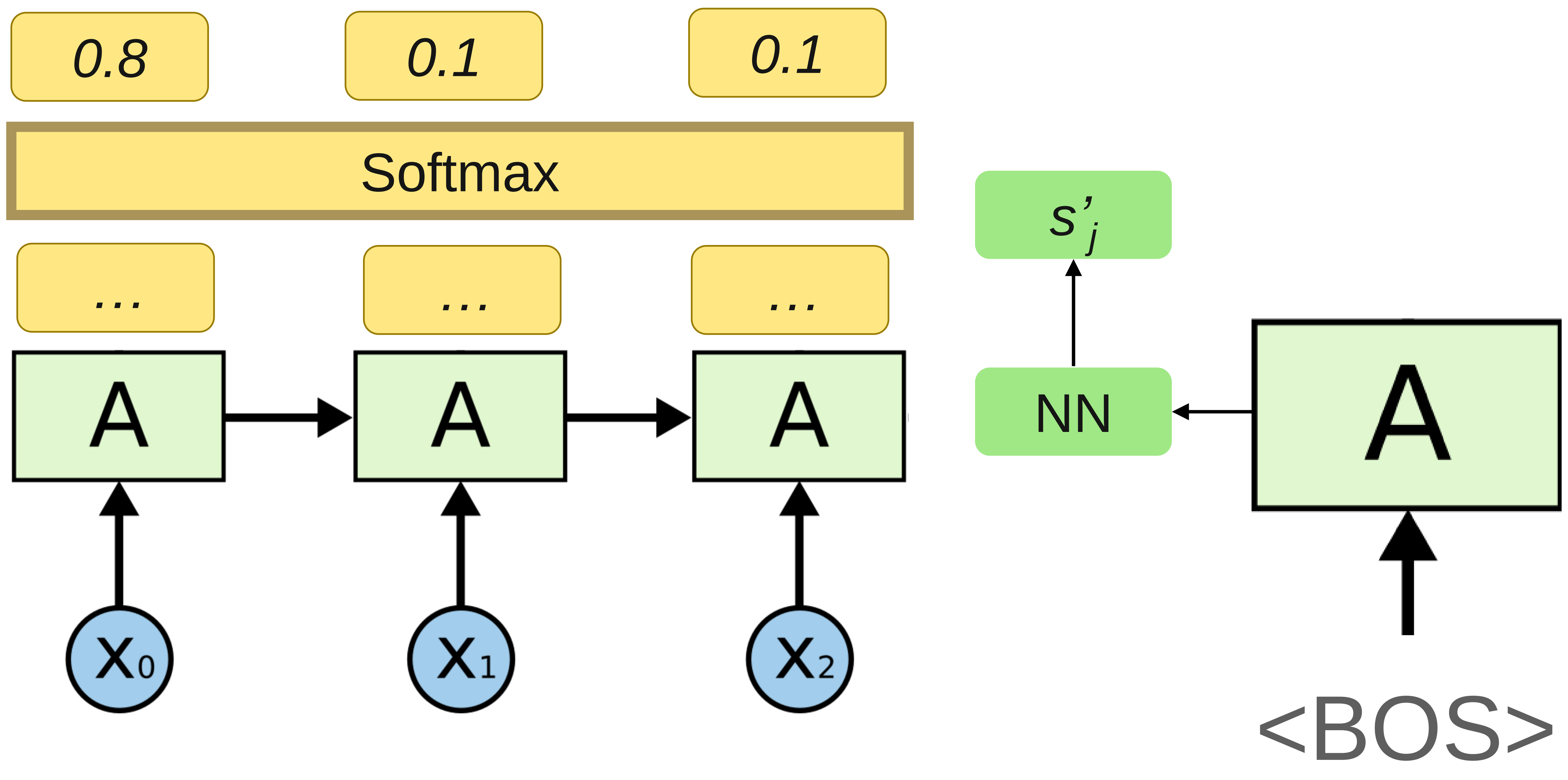
$$e_{ij} = v_a^{\mathrm{T}} \tanh(U_a s'_j + W_a h_i)$$



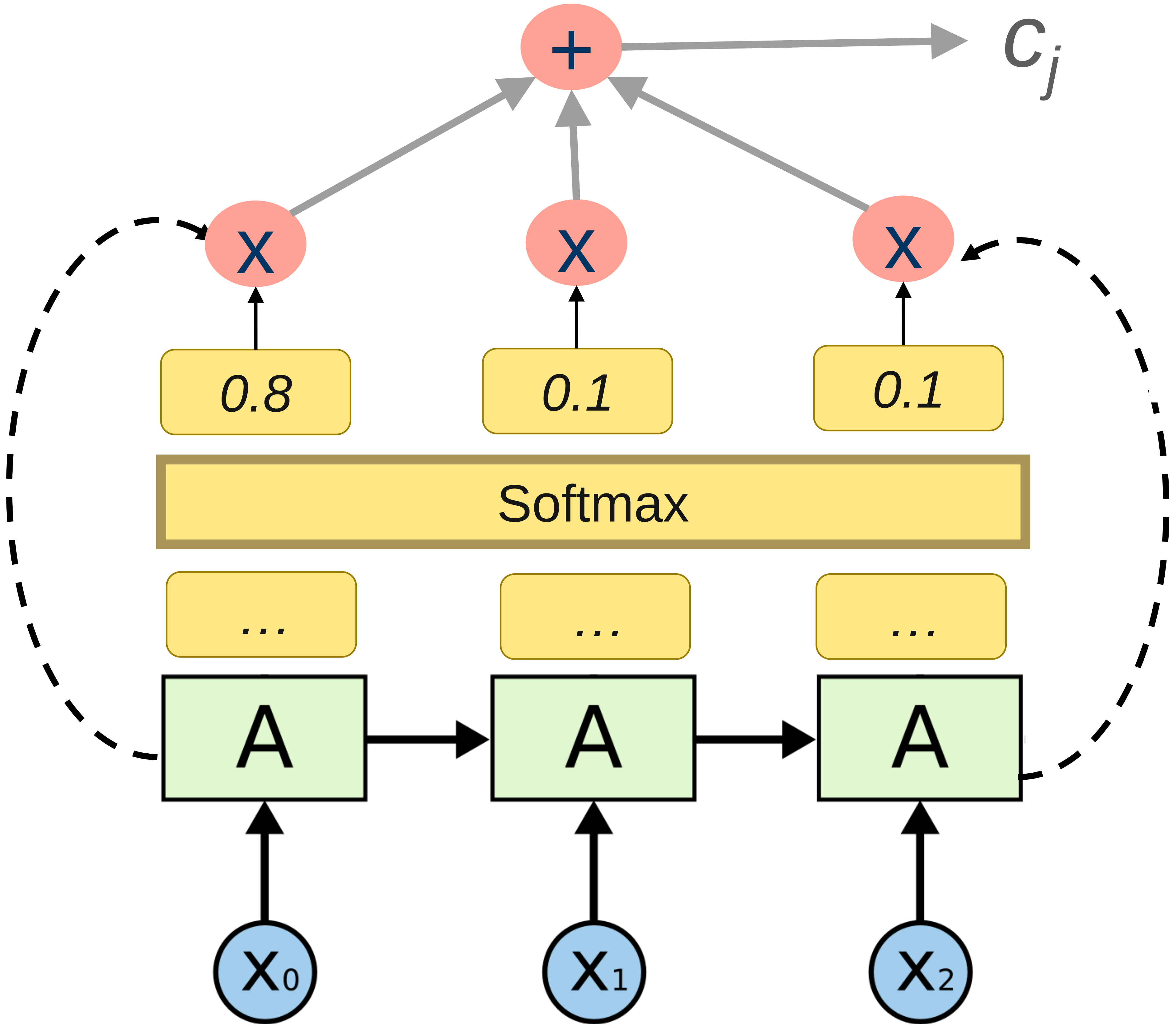
Attention – Step 3

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

$$e_{ij} = v_a^{\mathrm{T}} \tanh(U_a s'_j + W_a h_i)$$



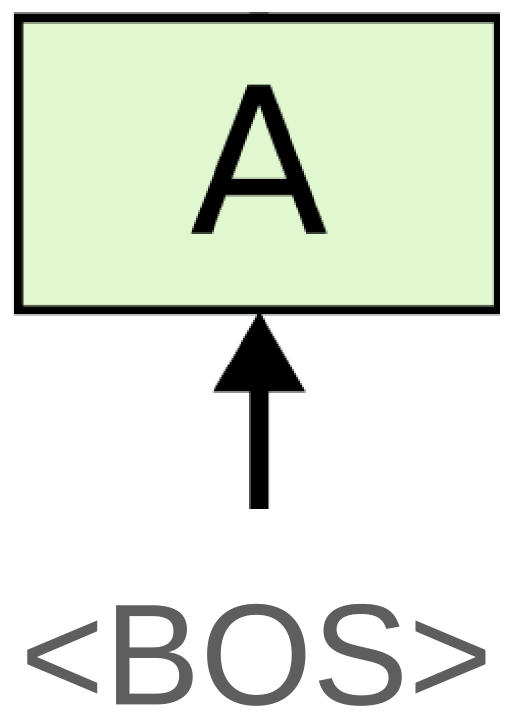
Attention – Step 4



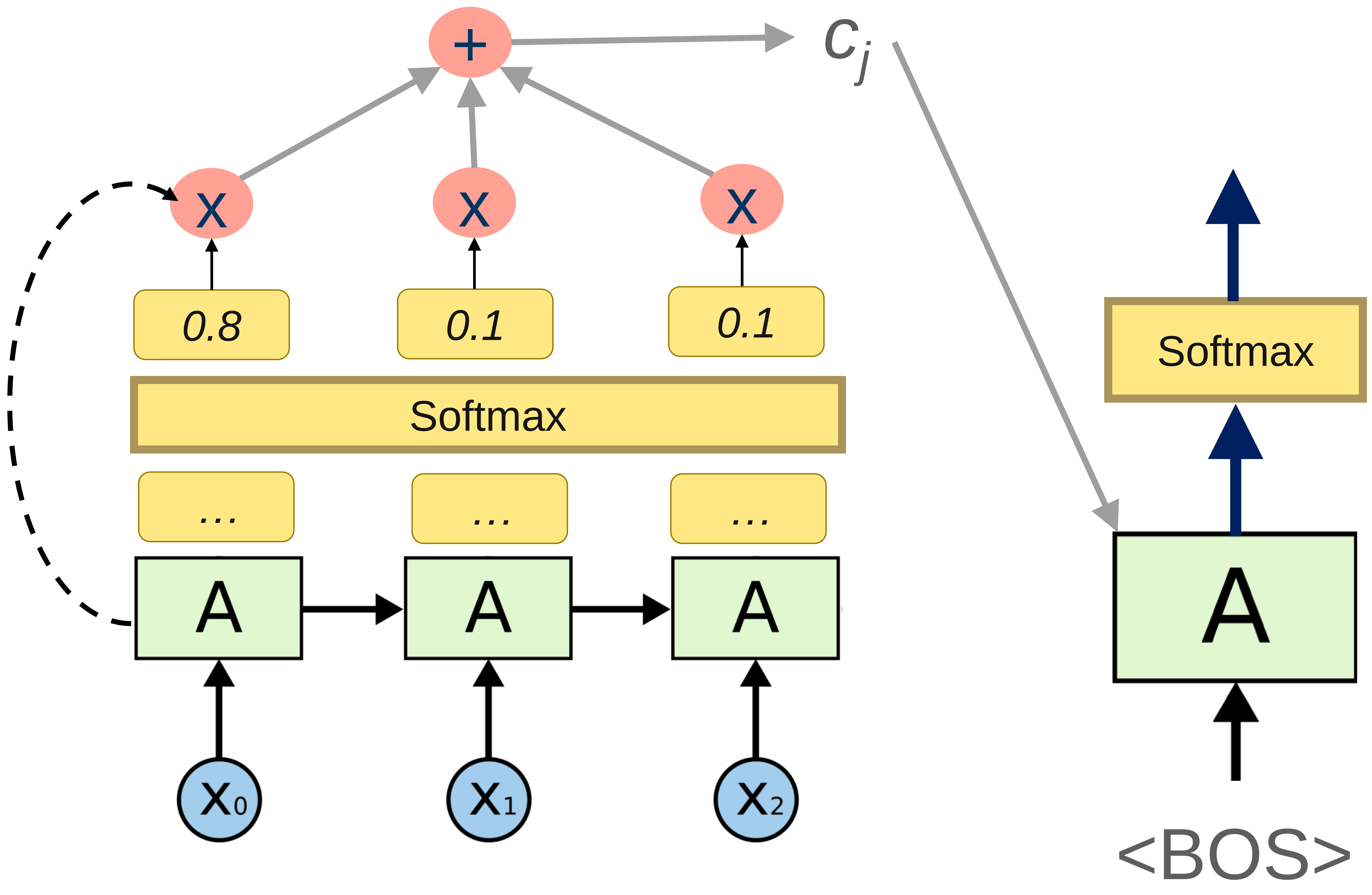
$$c_j = \mathrm{ATT}(C, s'_j) = \sum_{i=1}^{I} \alpha_{ij} h_i,$$

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

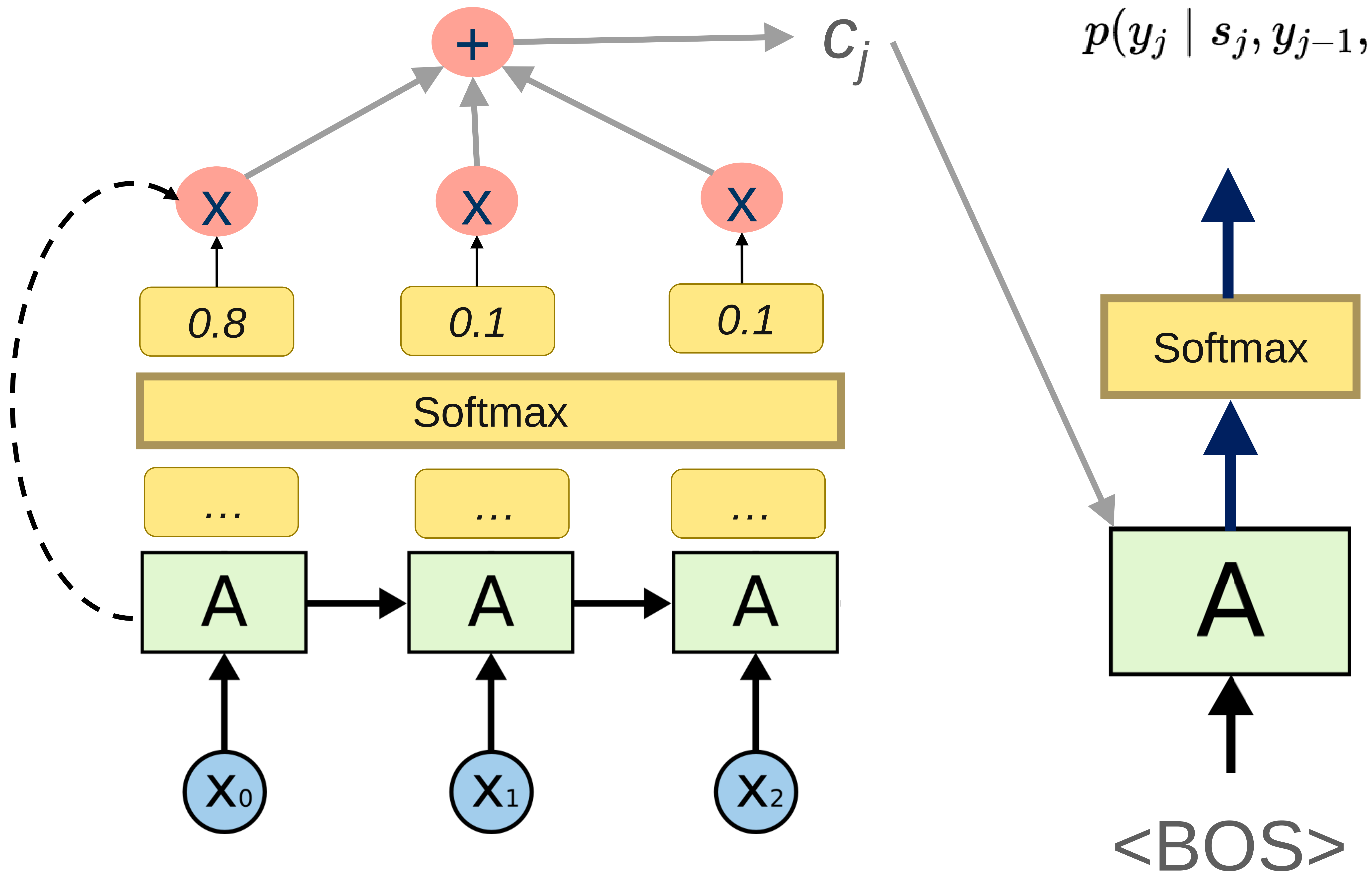
$$e_{ij} = v_a^{\mathrm{T}} \tanh(U_a s'_j + W_a h_i)$$



Attention – Step 5



Attention



$$p(y_j \mid s_j, y_{j-1}, c_j) = \mathrm{softmax}(W_o t_j)$$
$$t_j = \tanh(W_{t1} s_j + W_{t2} \mathrm{E}[y_{j-1}] + W_{t3} c_j)$$

RNNsearch (Bahdanau et al., 2015)

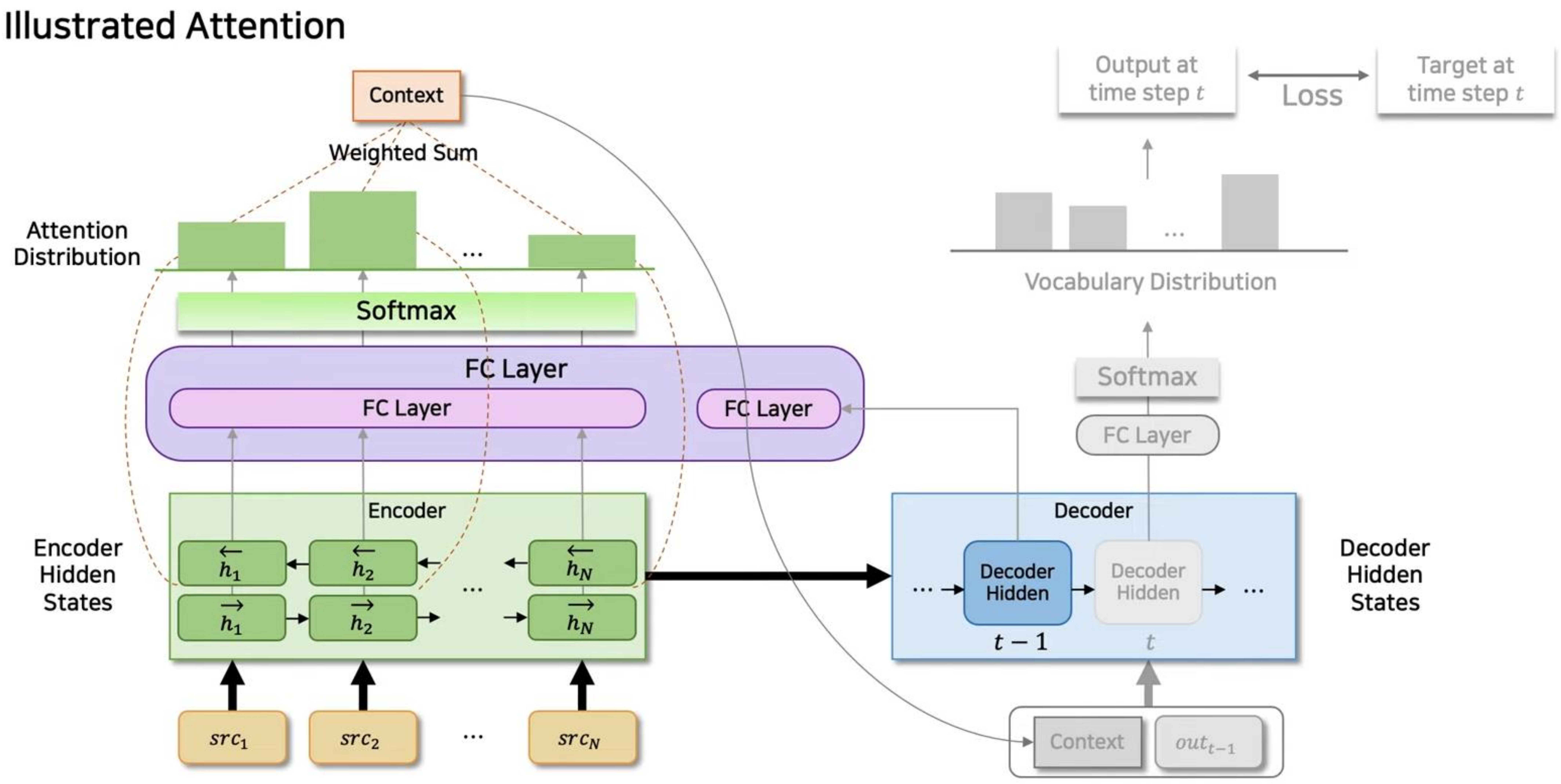
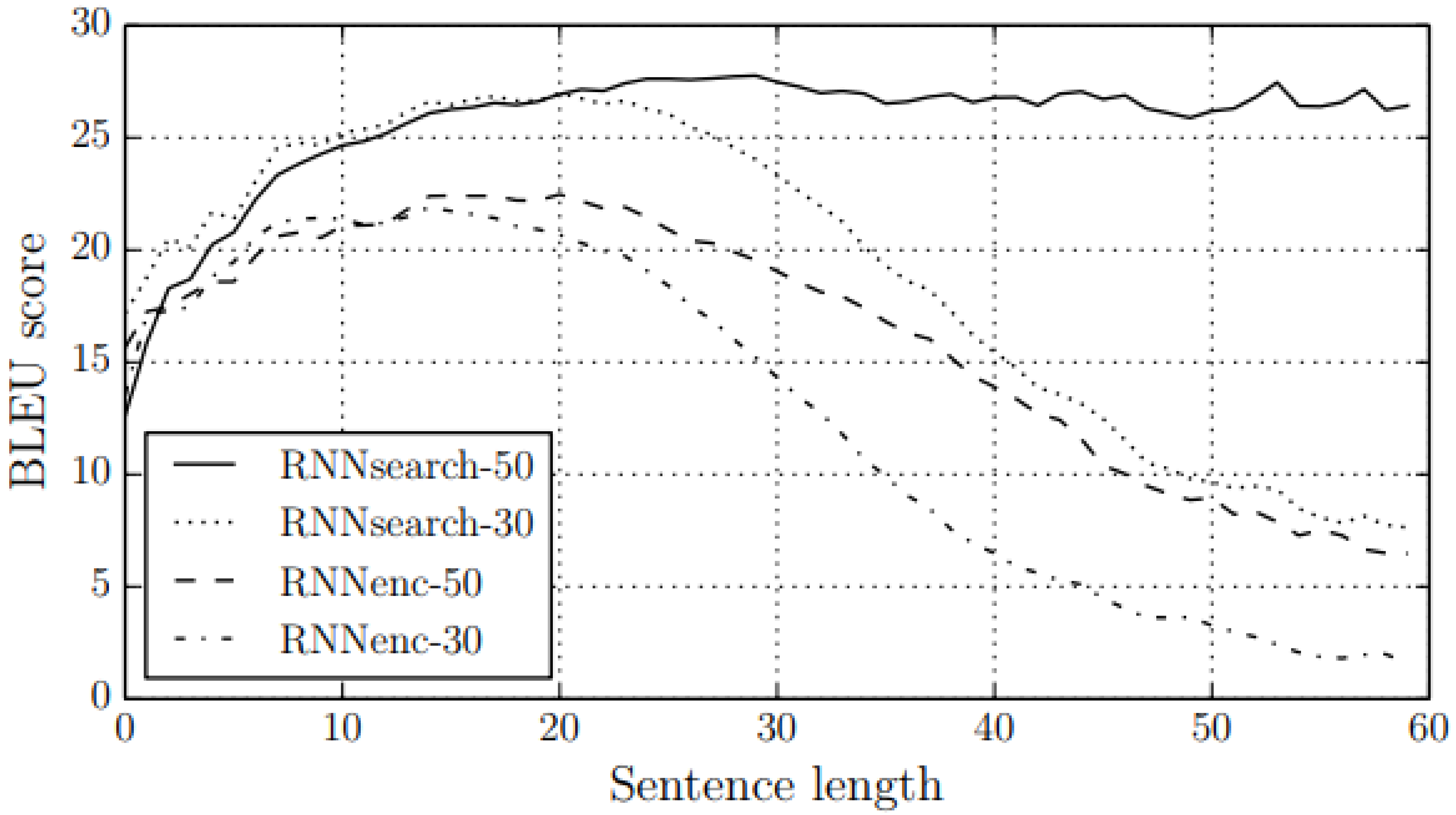


Image taken from : https://www.youtube.com/watch?v=S2msiG9g7Us

Results

Neural Machine Translation by Jointly Learning to Align and Translate (Bahdanau et al., 2015)



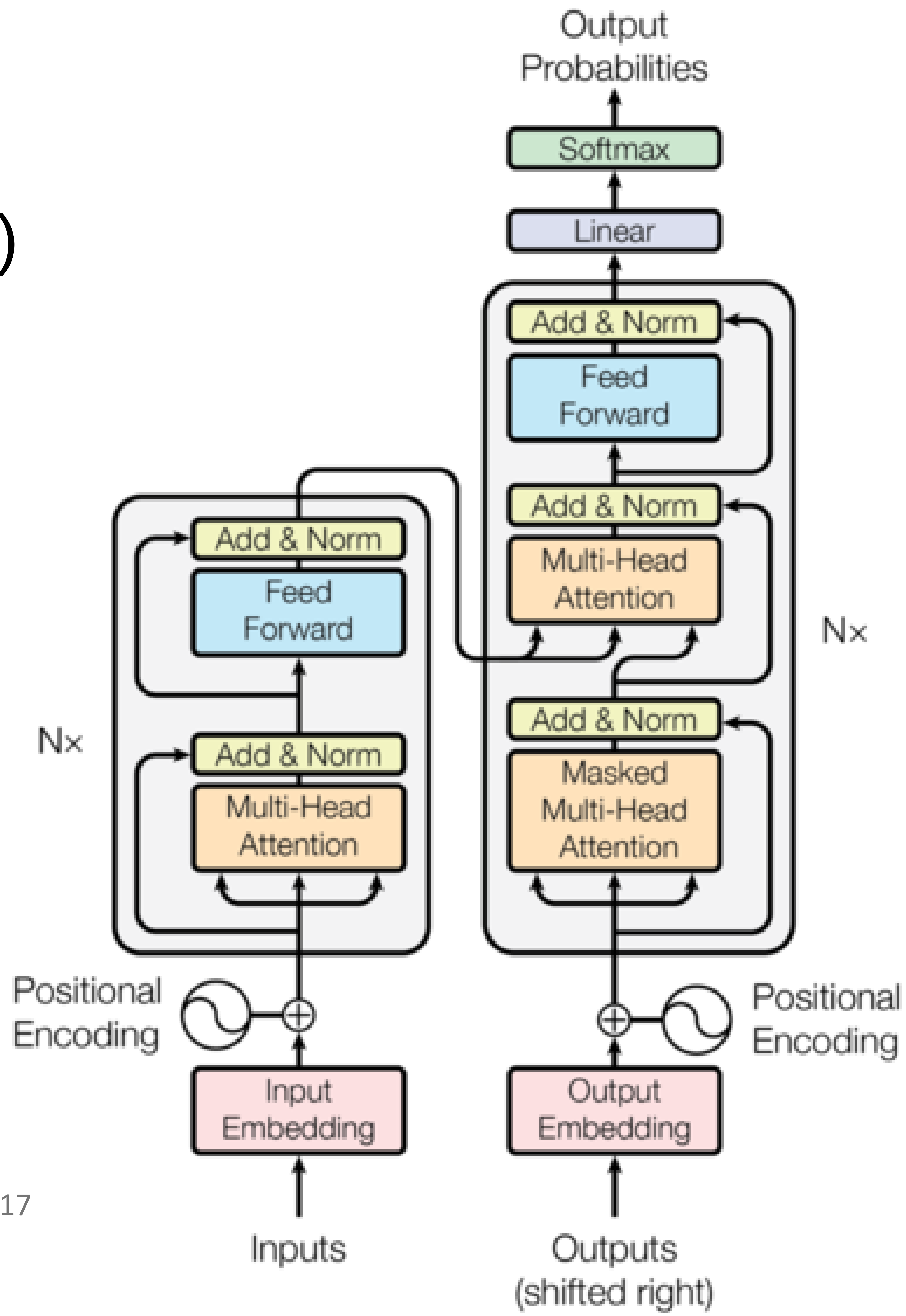
Summary

- Limitation of Seq2Seq model
- Attention Mechanism

Transformer

Transformer

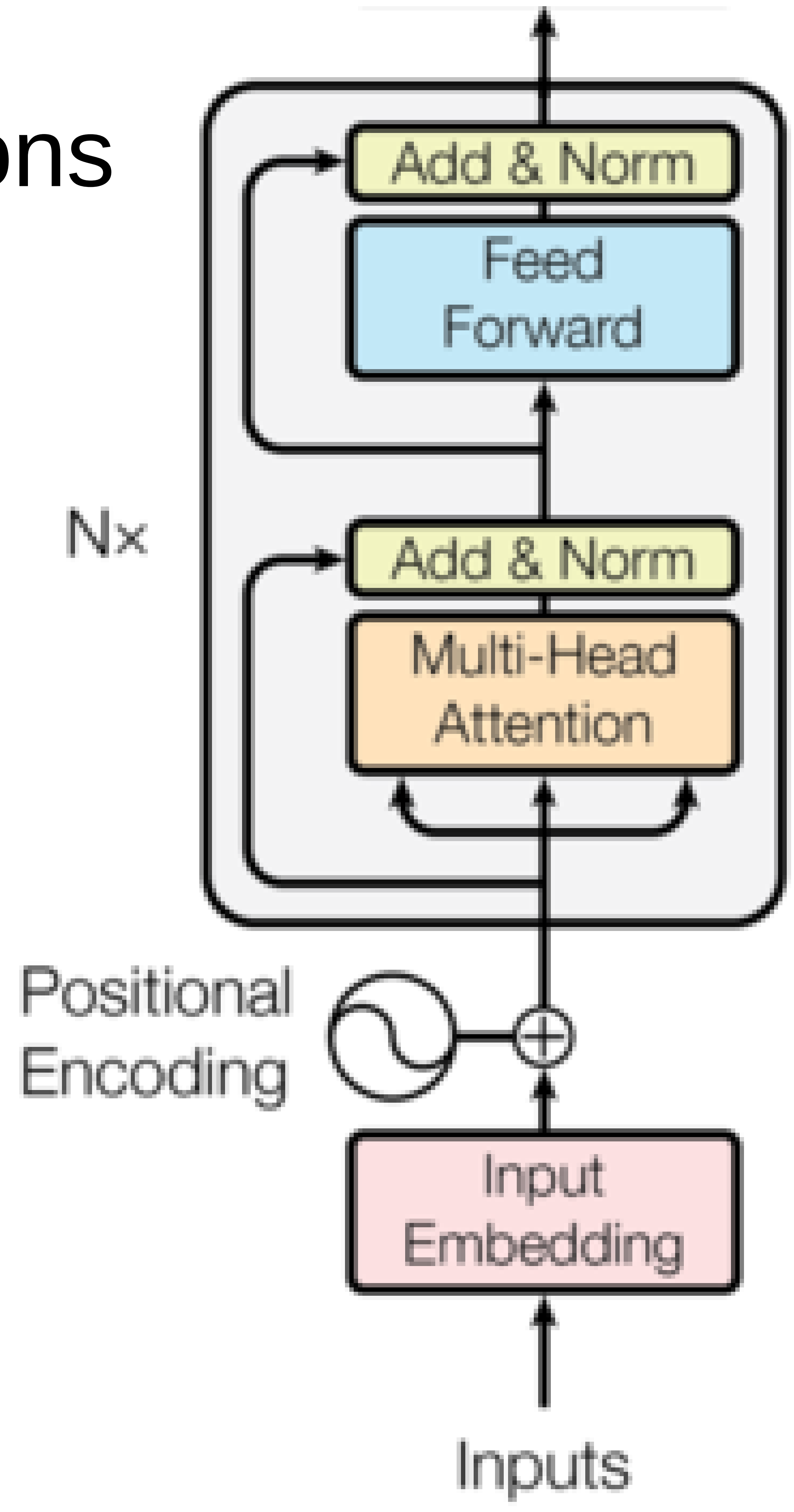
- Attention is all you need (Vaswani et al., 2017)
- An encoder-decoder framework for sequence-to-sequence modeling
- No recurrent units



From “Attention is all you need” paper by Vaswani, et al., 2017

Transformer Encoder

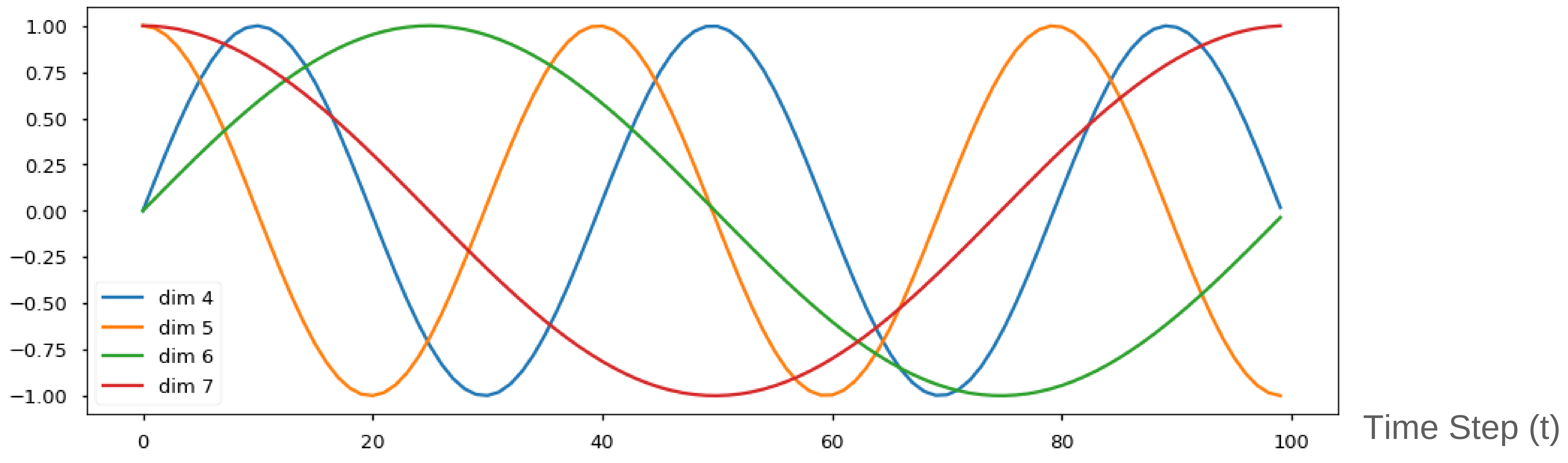
- N layers of Transformer blocks with residual connections
- Parameters in each layer are not shared
- Input tokens are processed in parallel

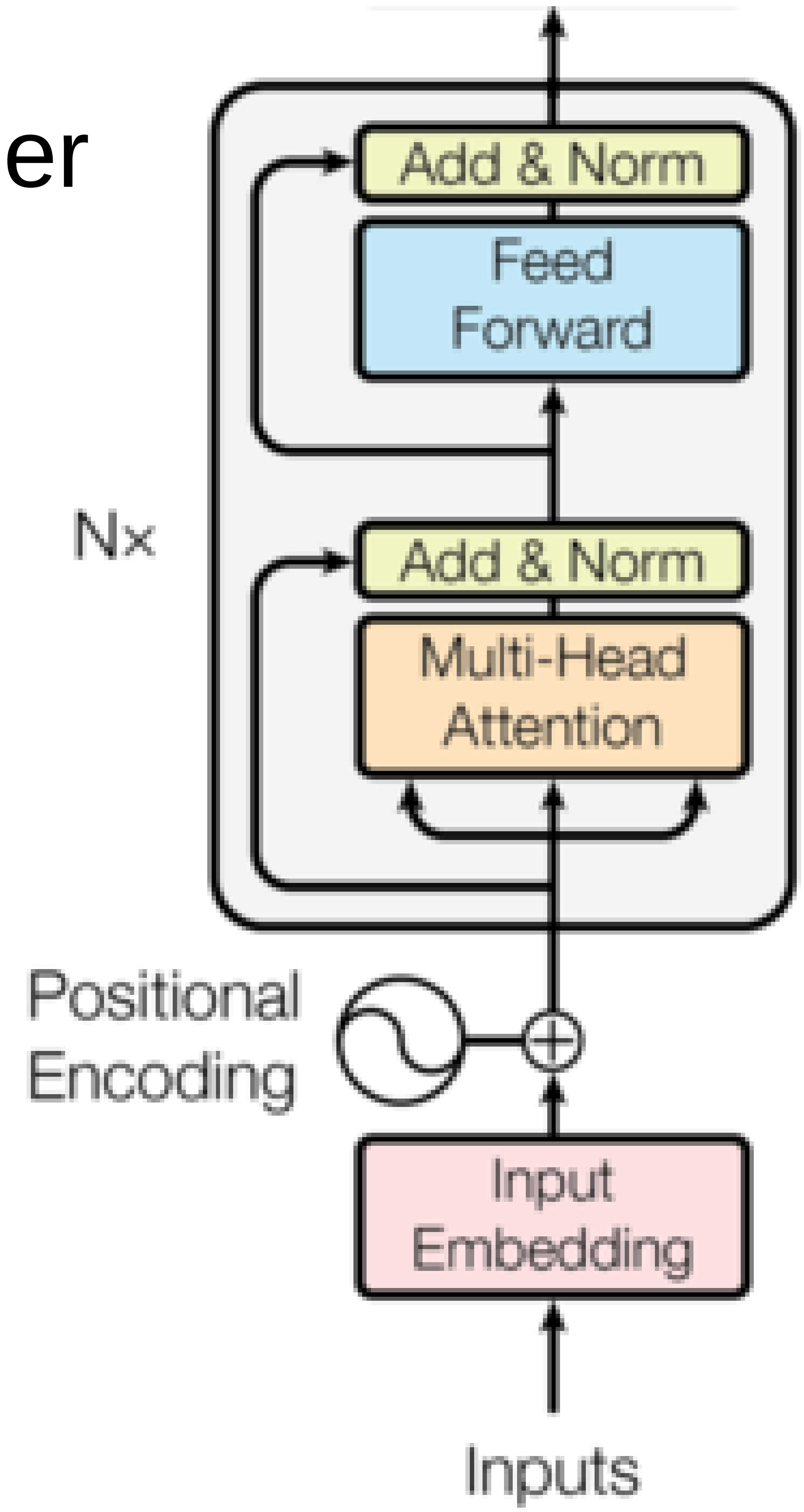


Positional Encoding

- Positional encoding provide the model about word order
- Sinusoidal position encoding

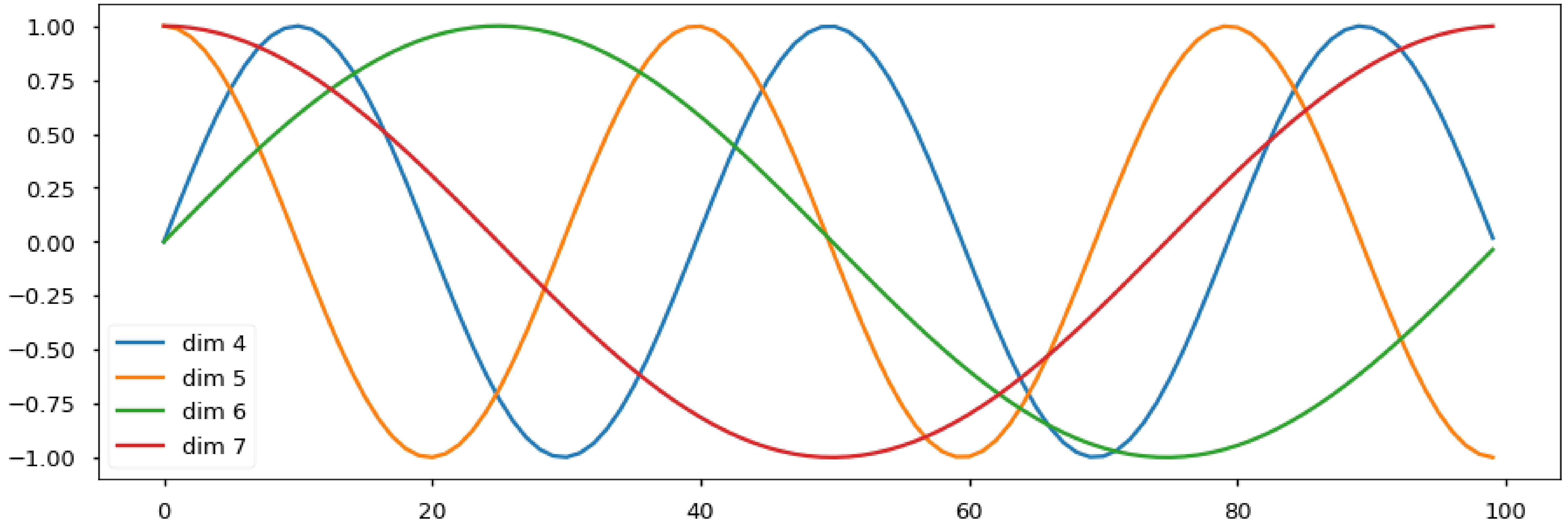
$$\mathbf{x}'_t = \mathbf{W}_{\text{emb}}(\mathbf{x}_t) + \vec{p}_t$$





Positional Encoding

$$\mathbf{x}'_t = \mathrm{W}_{\mathrm{emb}}\left(\mathbf{x}_t\right) + \vec{p}_t$$



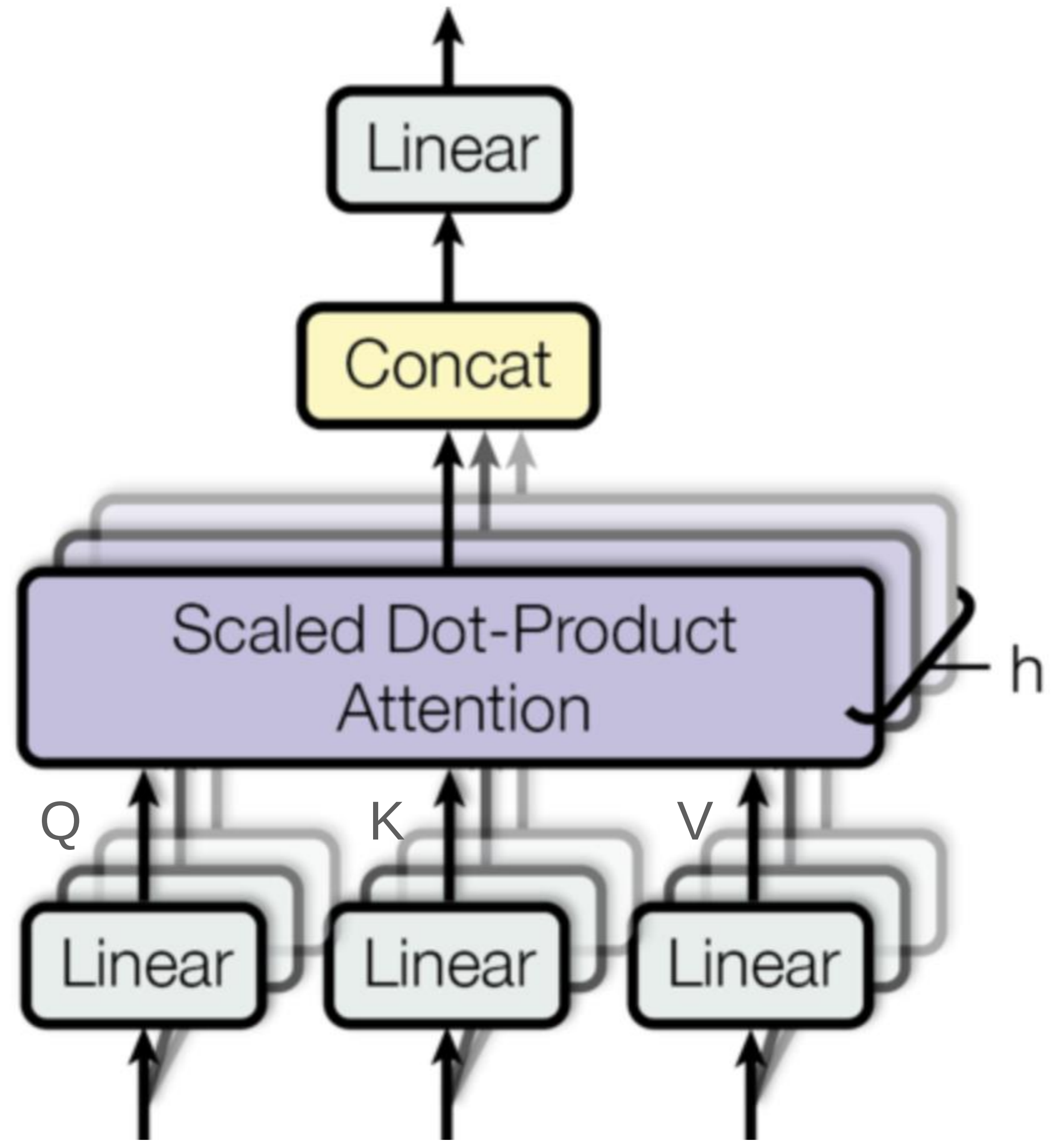
Time Step (t)

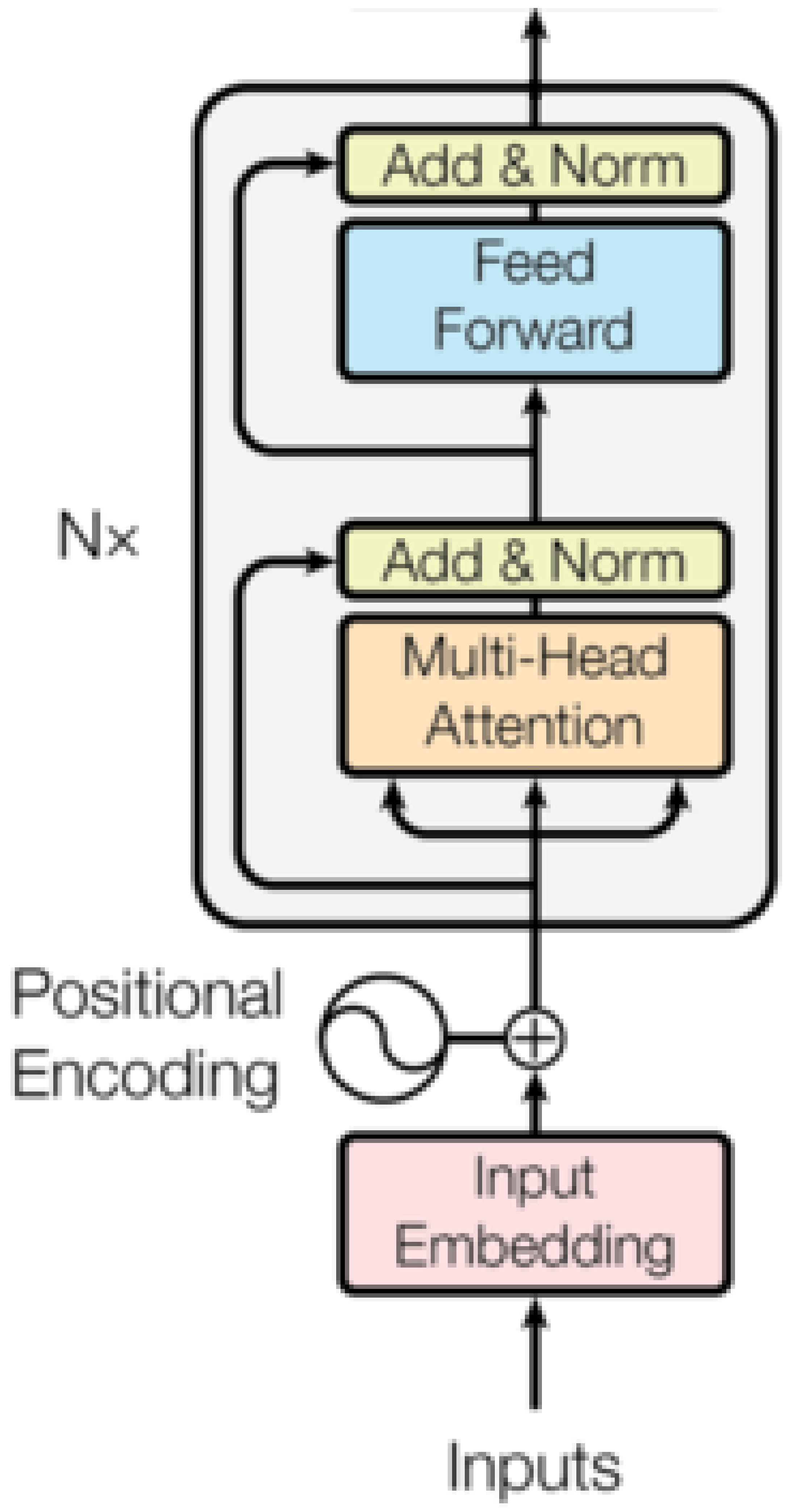
$$\vec{p_t} = \begin{bmatrix} \sin(\omega_1 . t) \\ \cos(\omega_1 . t) \\ \\ \sin(\omega_2 . t) \\ \cos(\omega_2 . t) \\ \\ \vdots \\ \\ \sin(\omega_{d/2} . t) \\ \cos(\omega_{d/2} . t) \end{bmatrix}_{d\times 1}$$

$$\omega_k = \frac{1}{10000^{2k/d}}$$

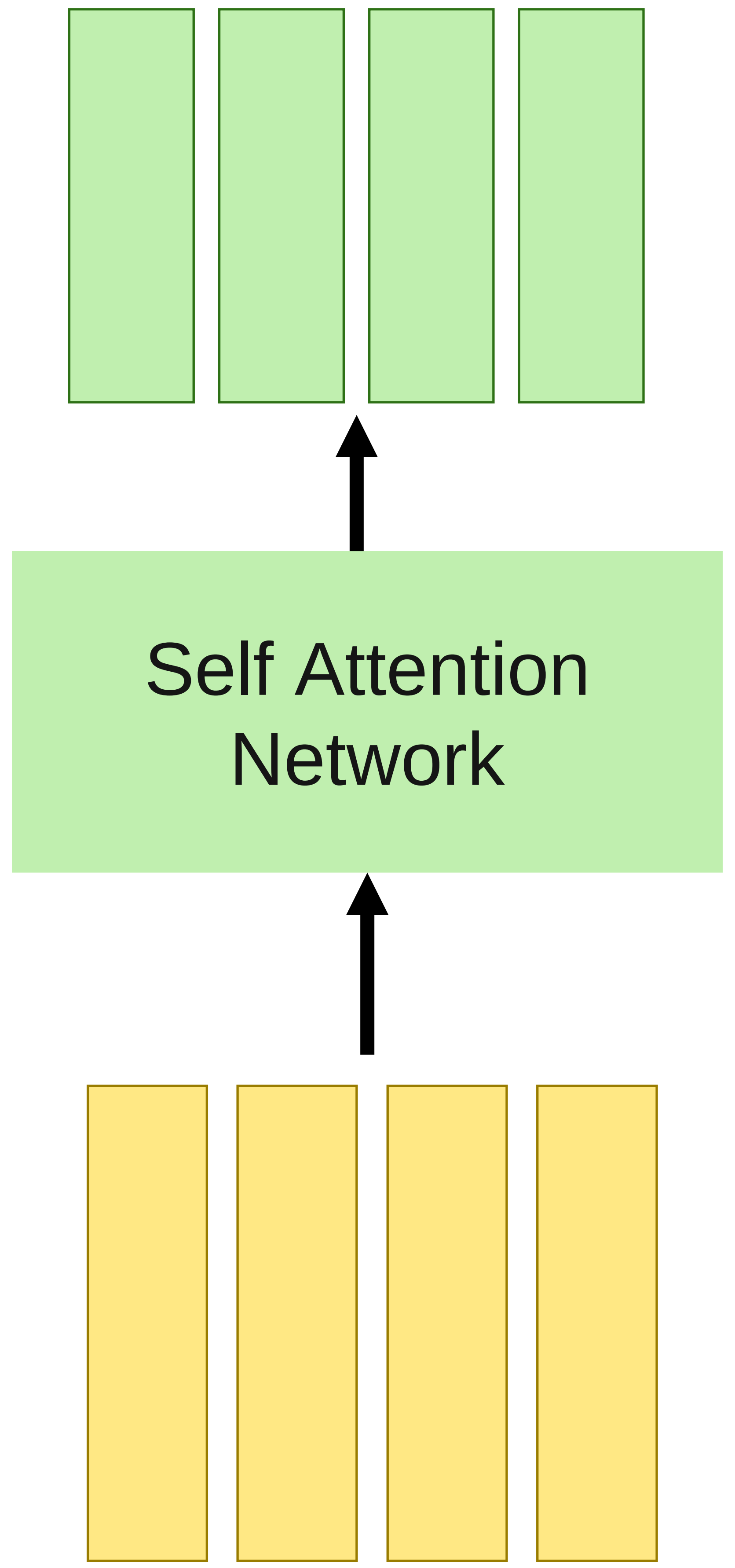
Multi-head Attention

- Multi-head Attention is the concatenation of the outputs from self-attention network (SAN)

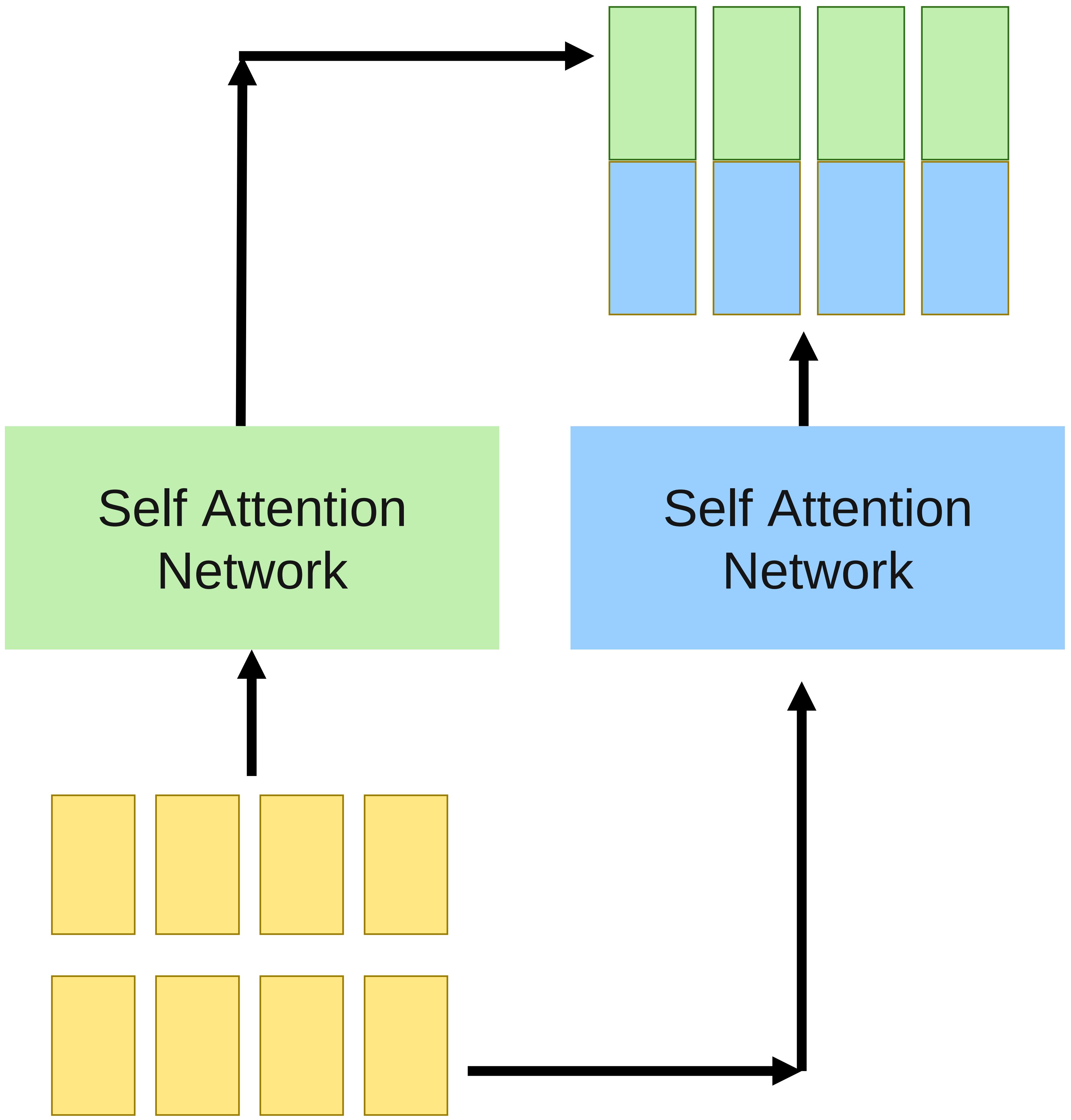




Multi-head Attention



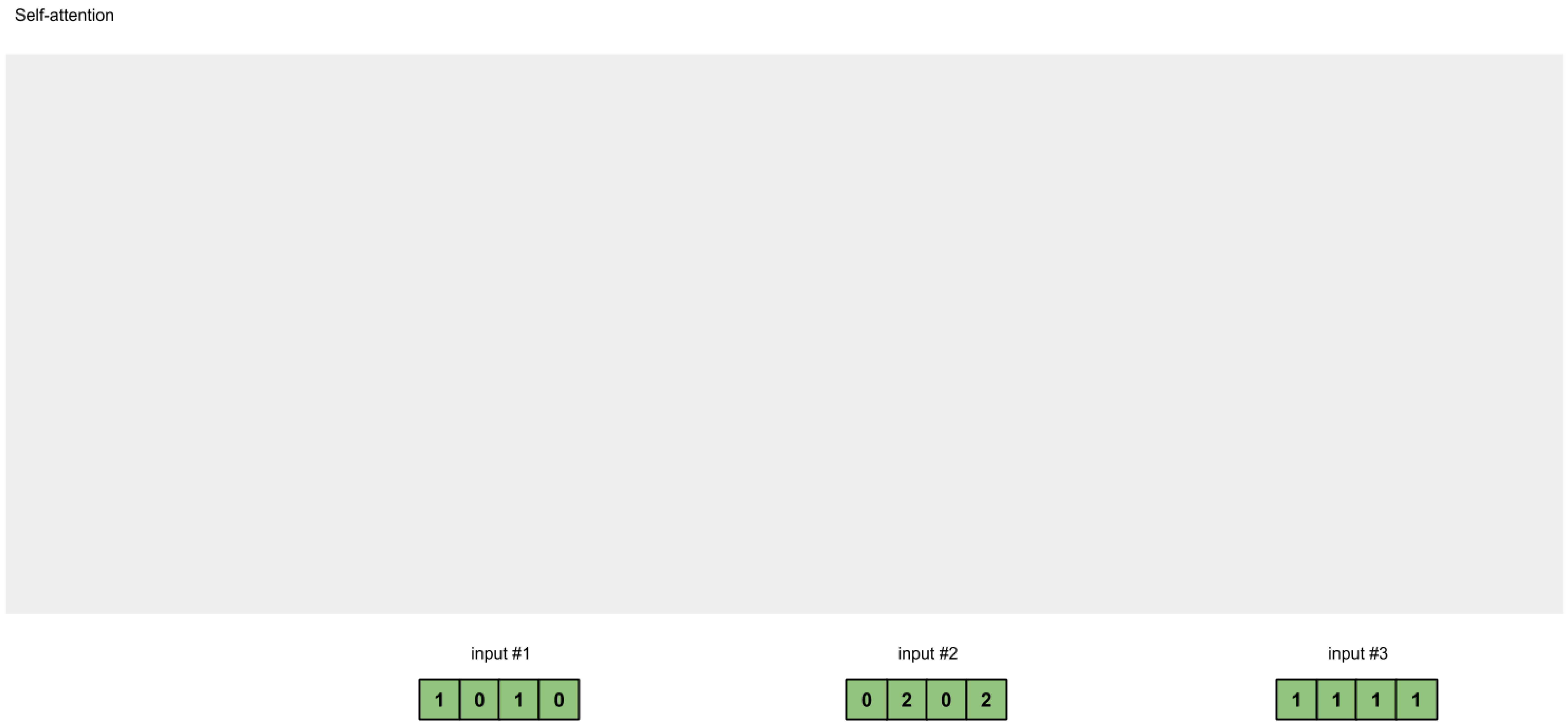
Single head self-attention (m=1)



Multi head self-attention (m=2)

Self-Attention Network

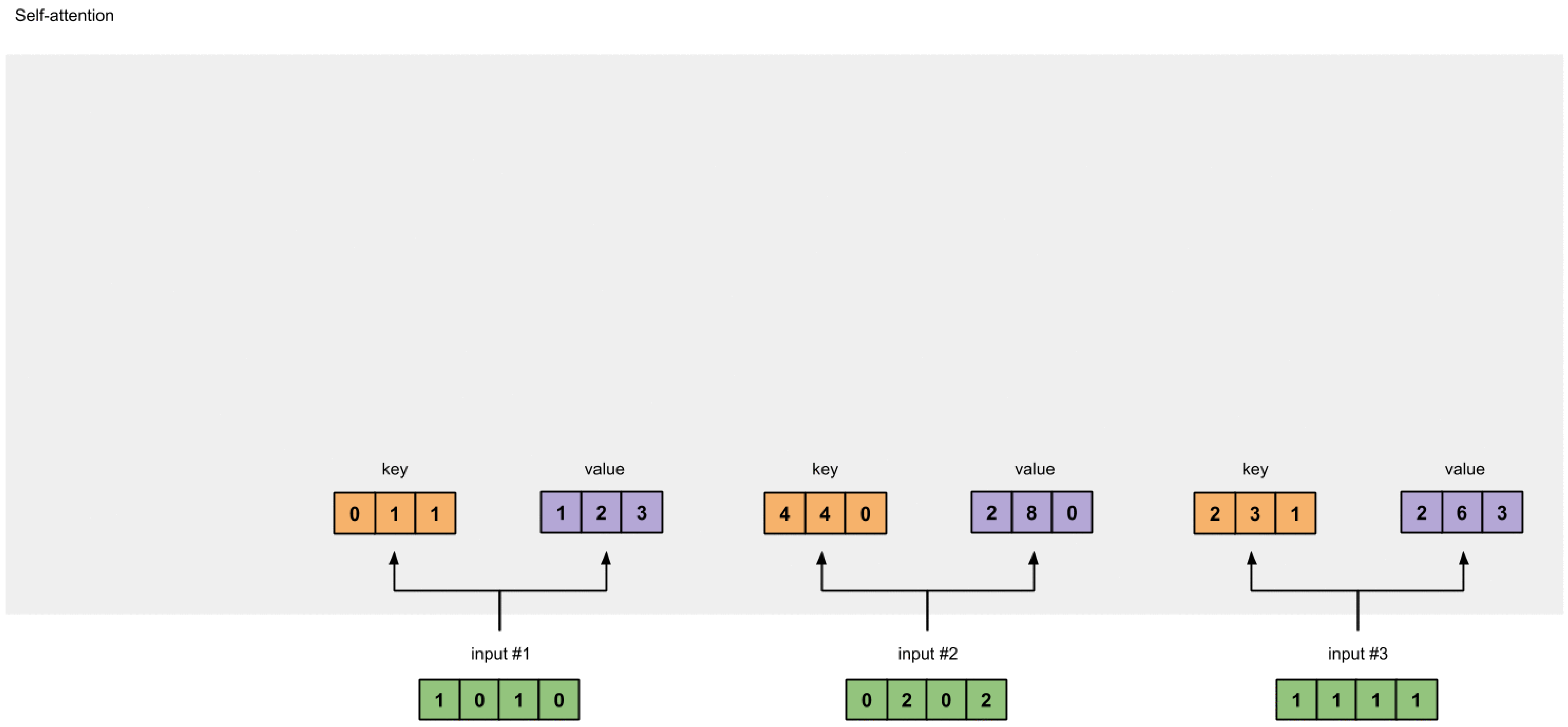
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a

Self-Attention Network

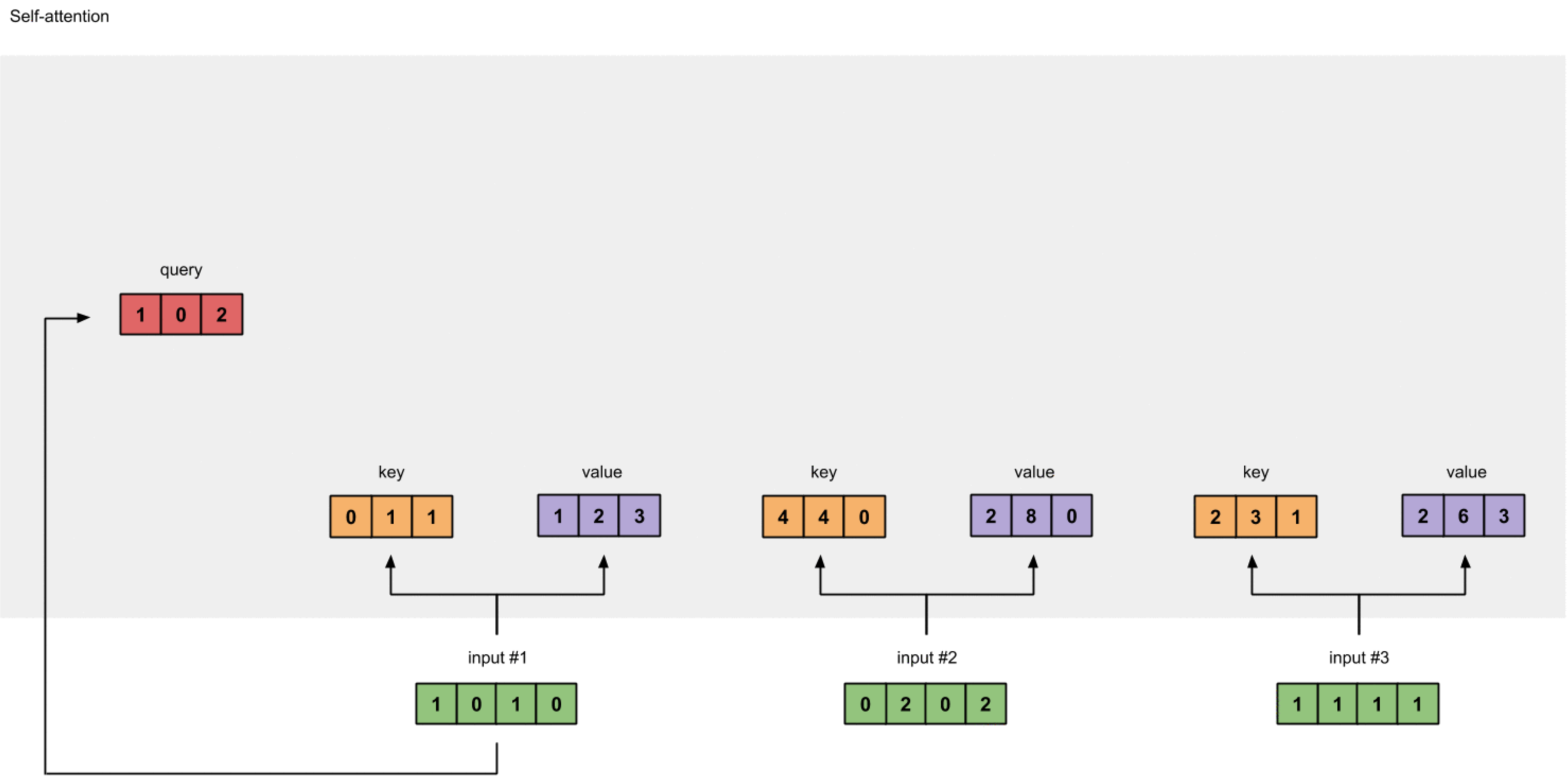
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a

Self-Attention Network

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

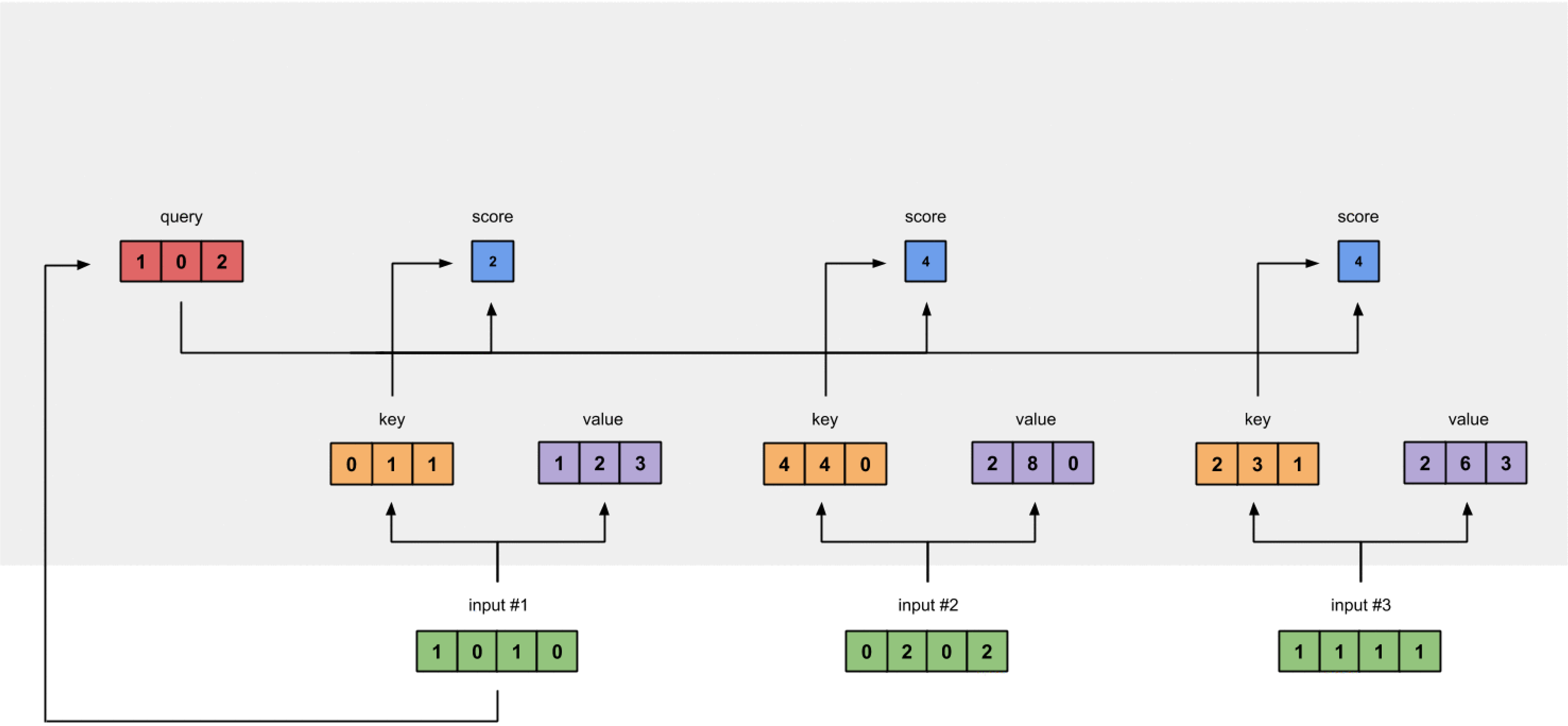


https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a

Self-Attention Network

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

Self-attention

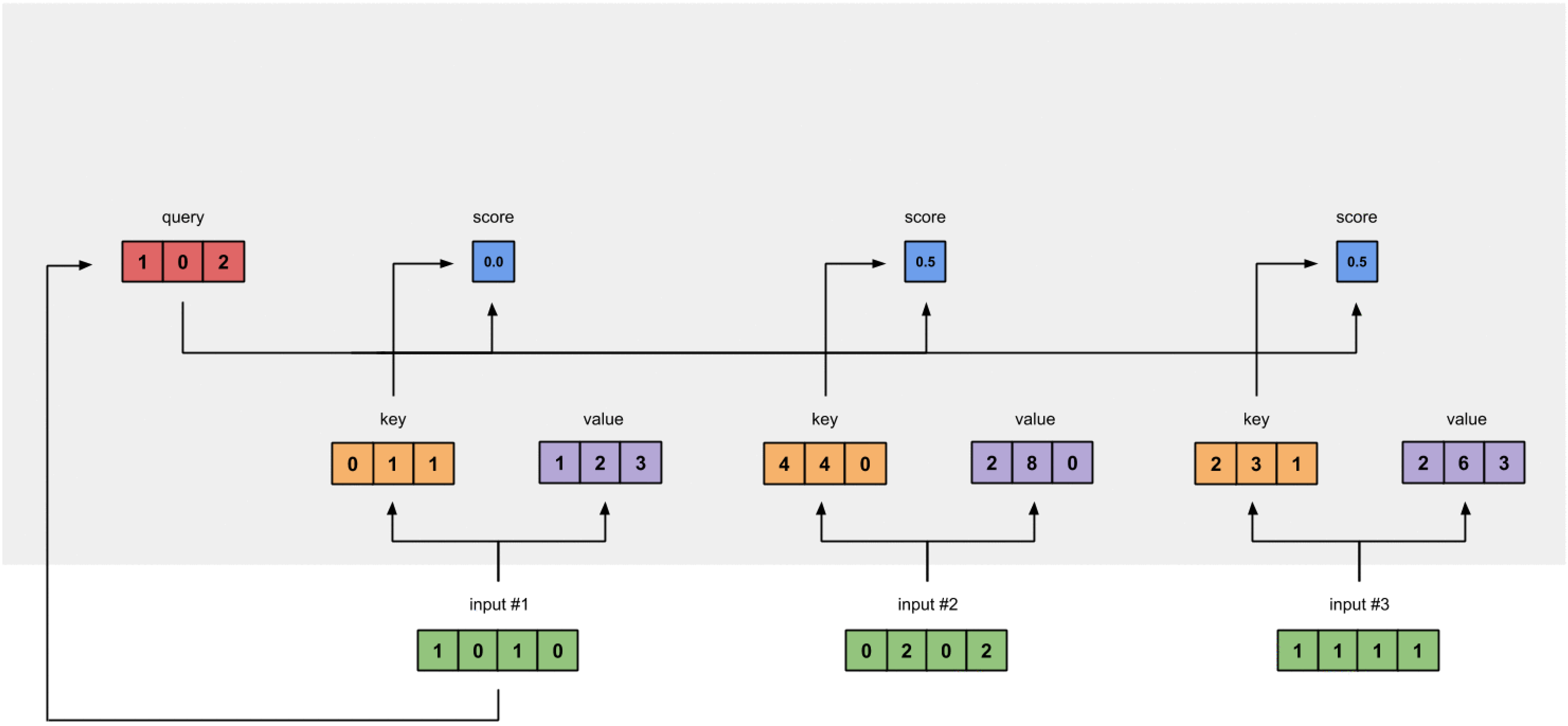


https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a

Self-Attention Network

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

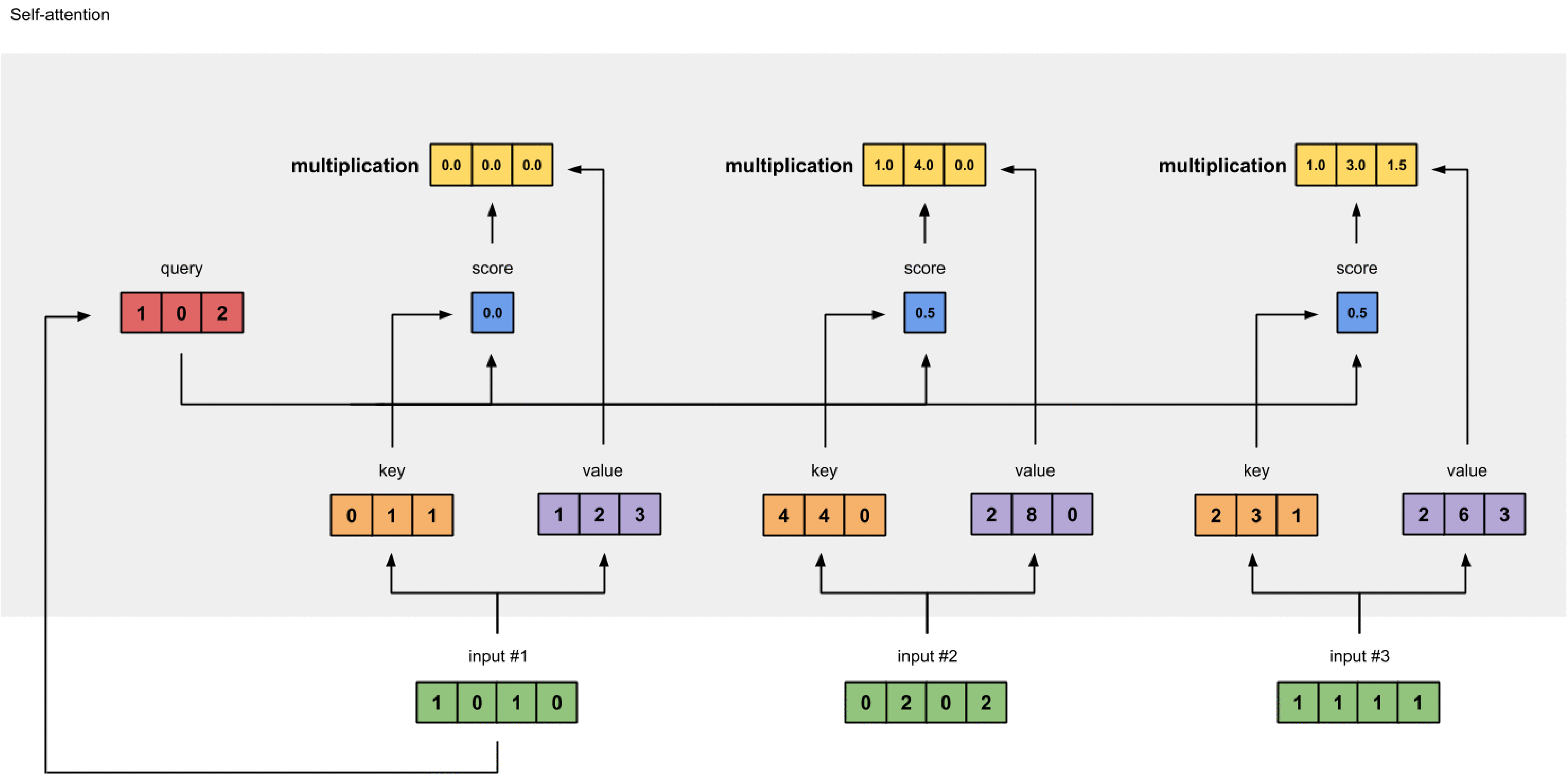
Self-attention



https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a

Self-Attention Network

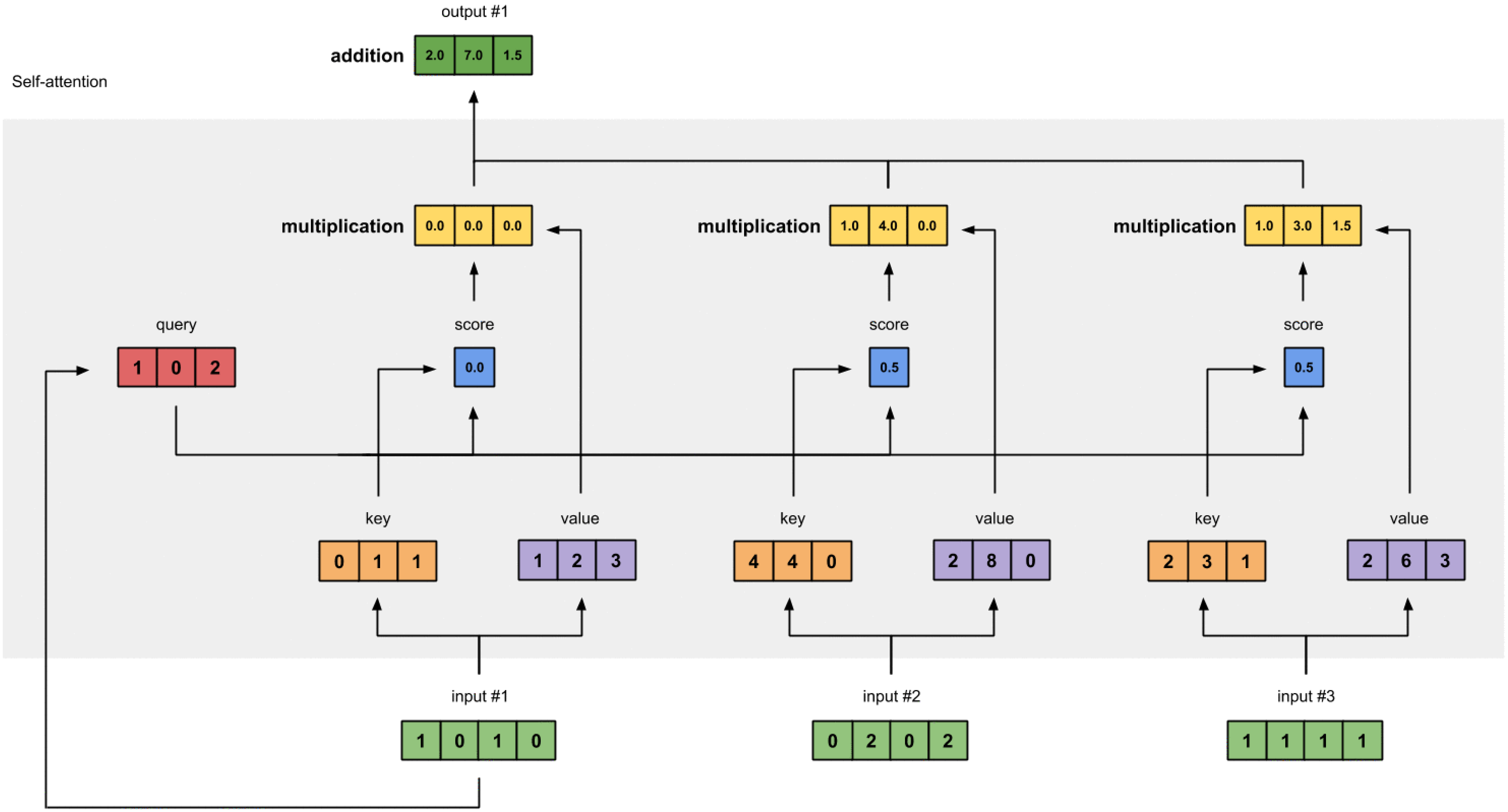
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$



https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a

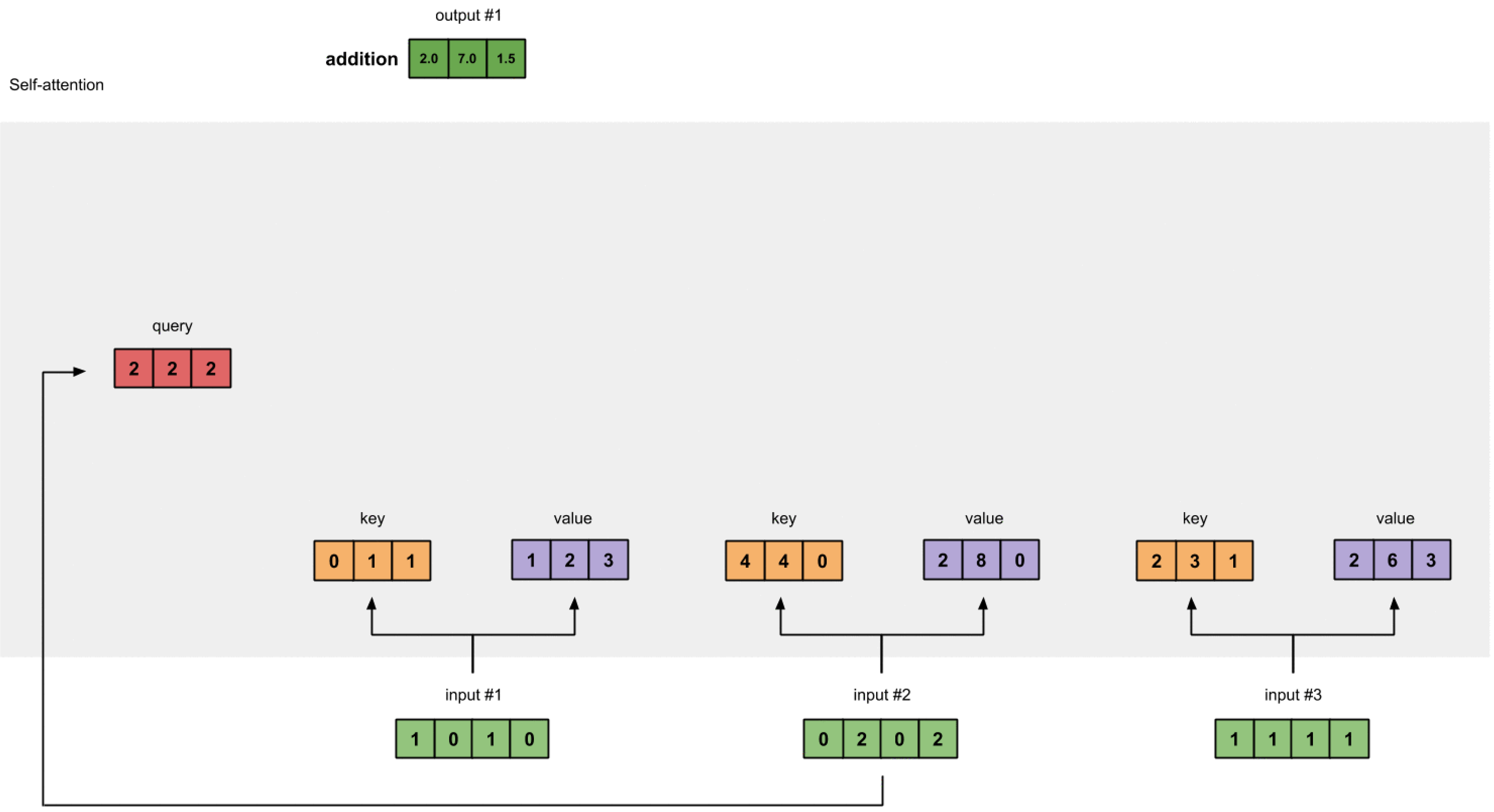
Self-Attention Network

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



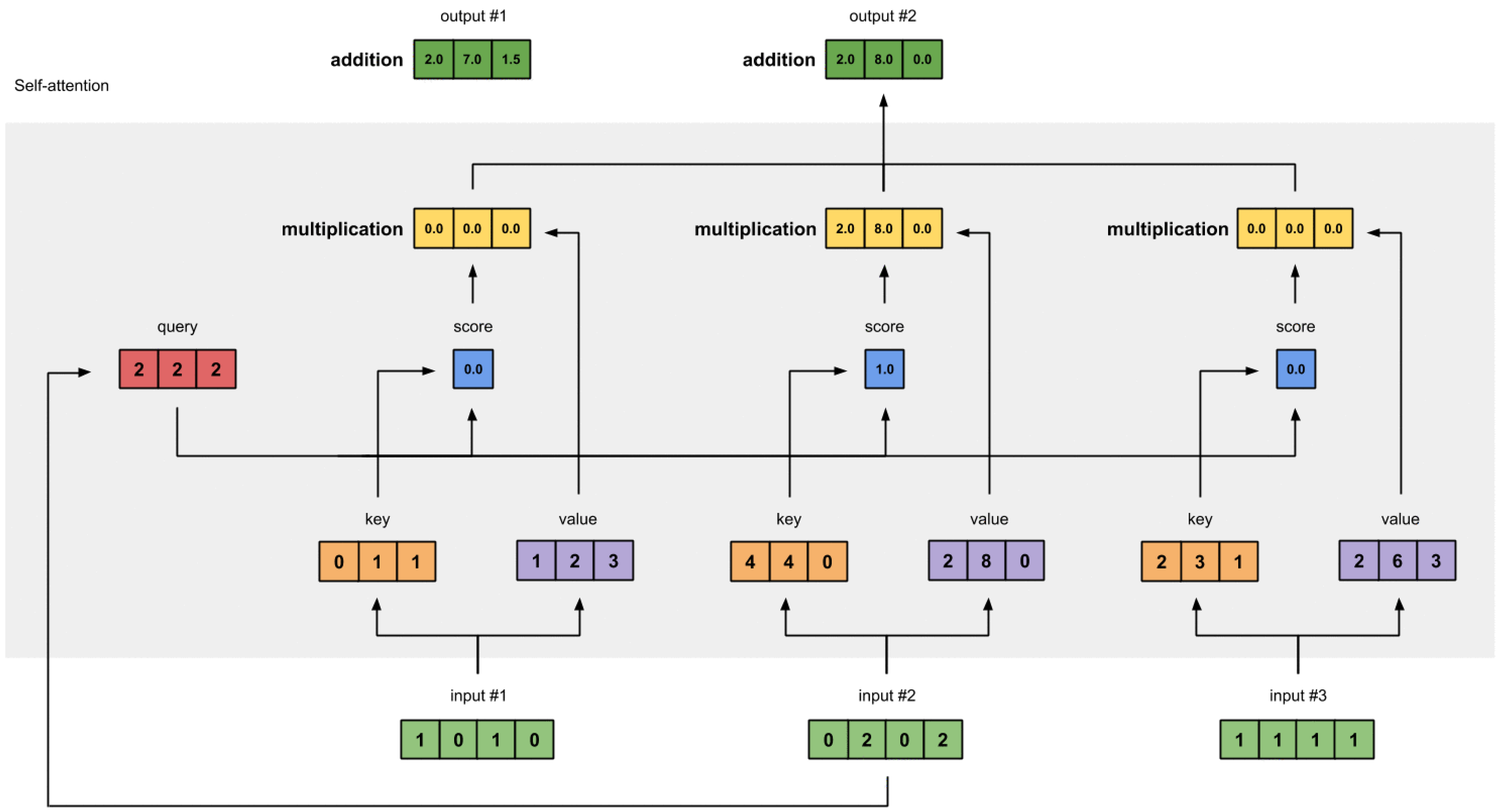
https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a

Self-Attention Network



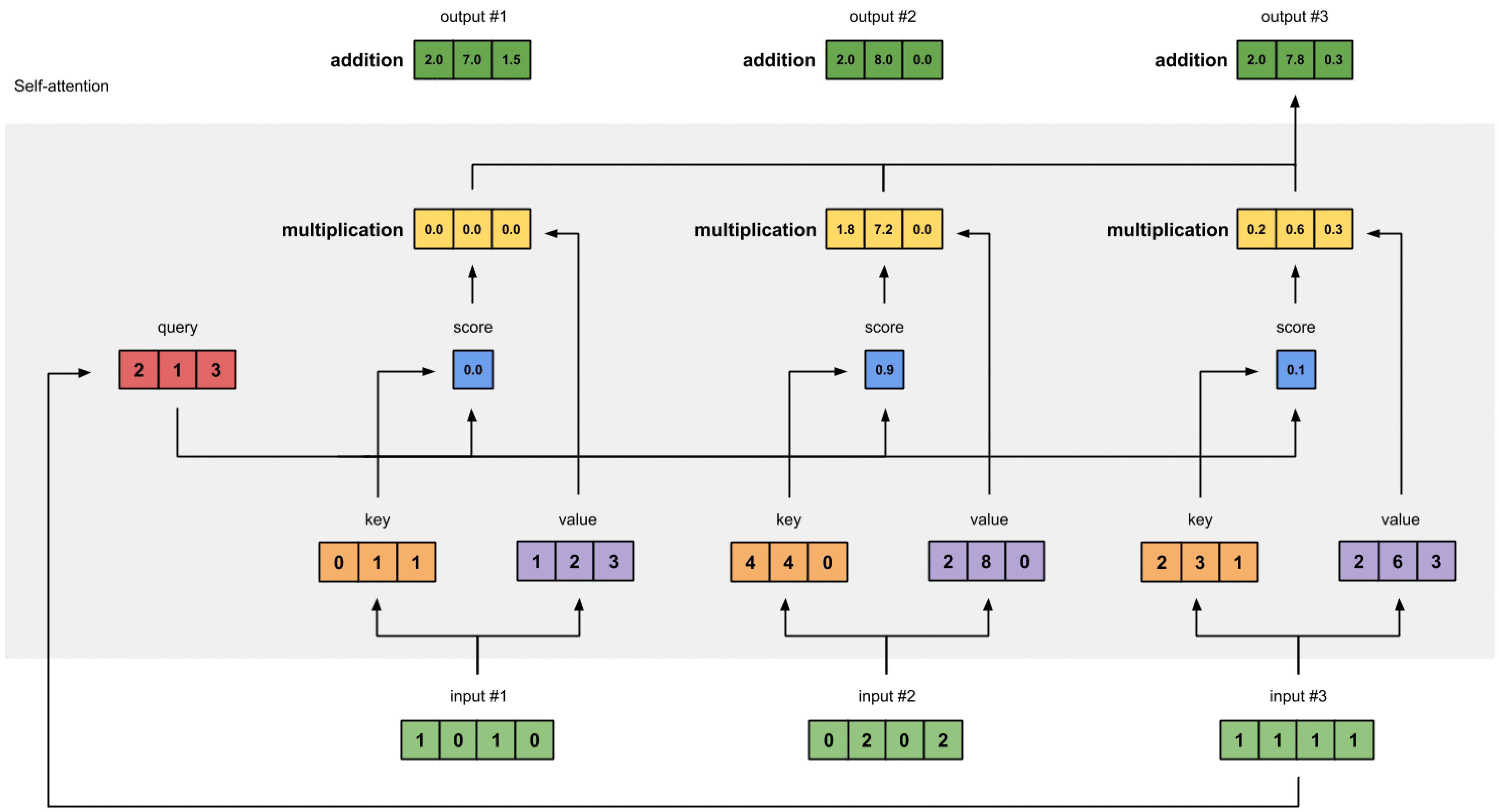
https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a

Self-Attention Network



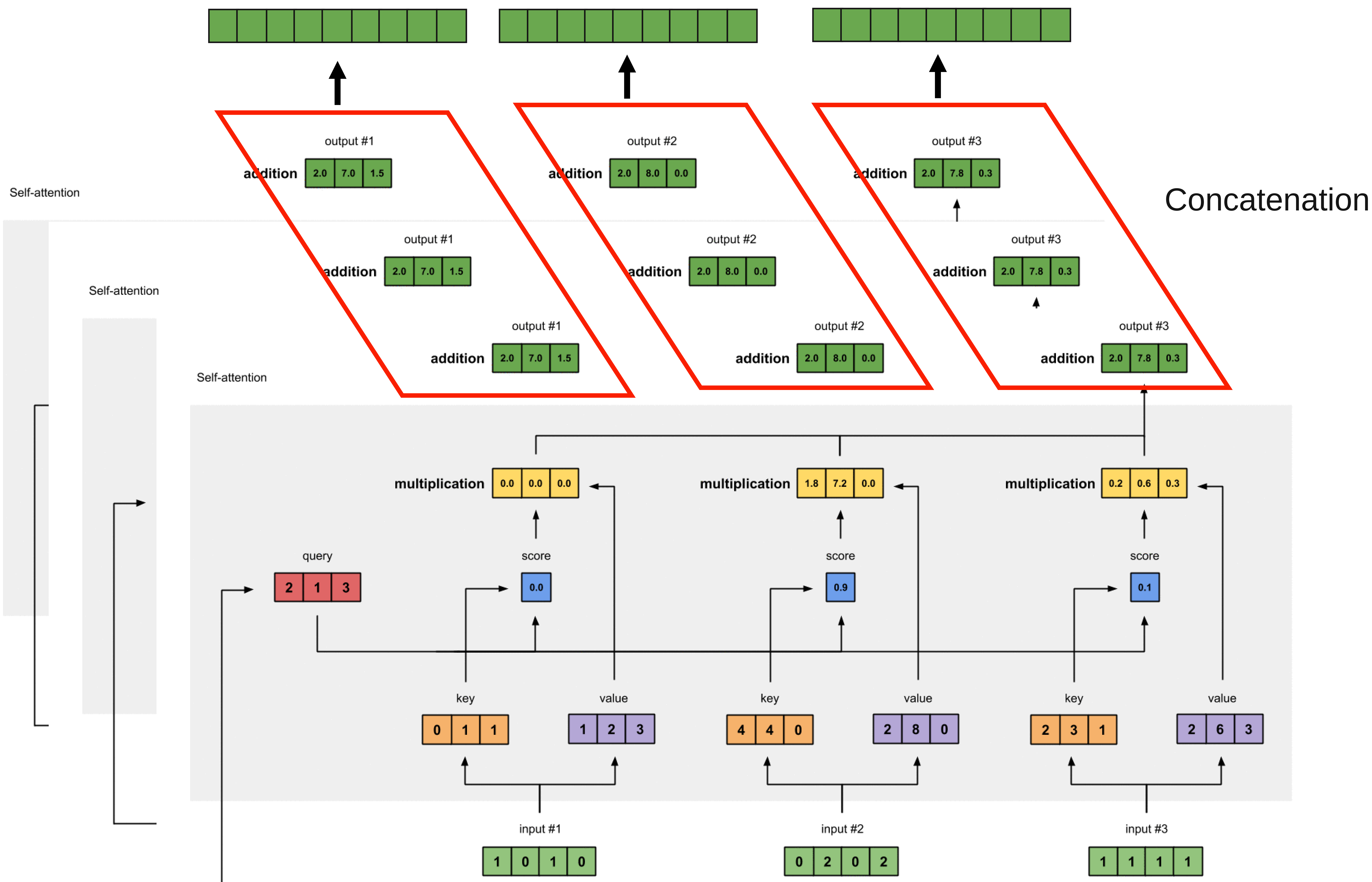
https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a

Self-Attention Network



https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a

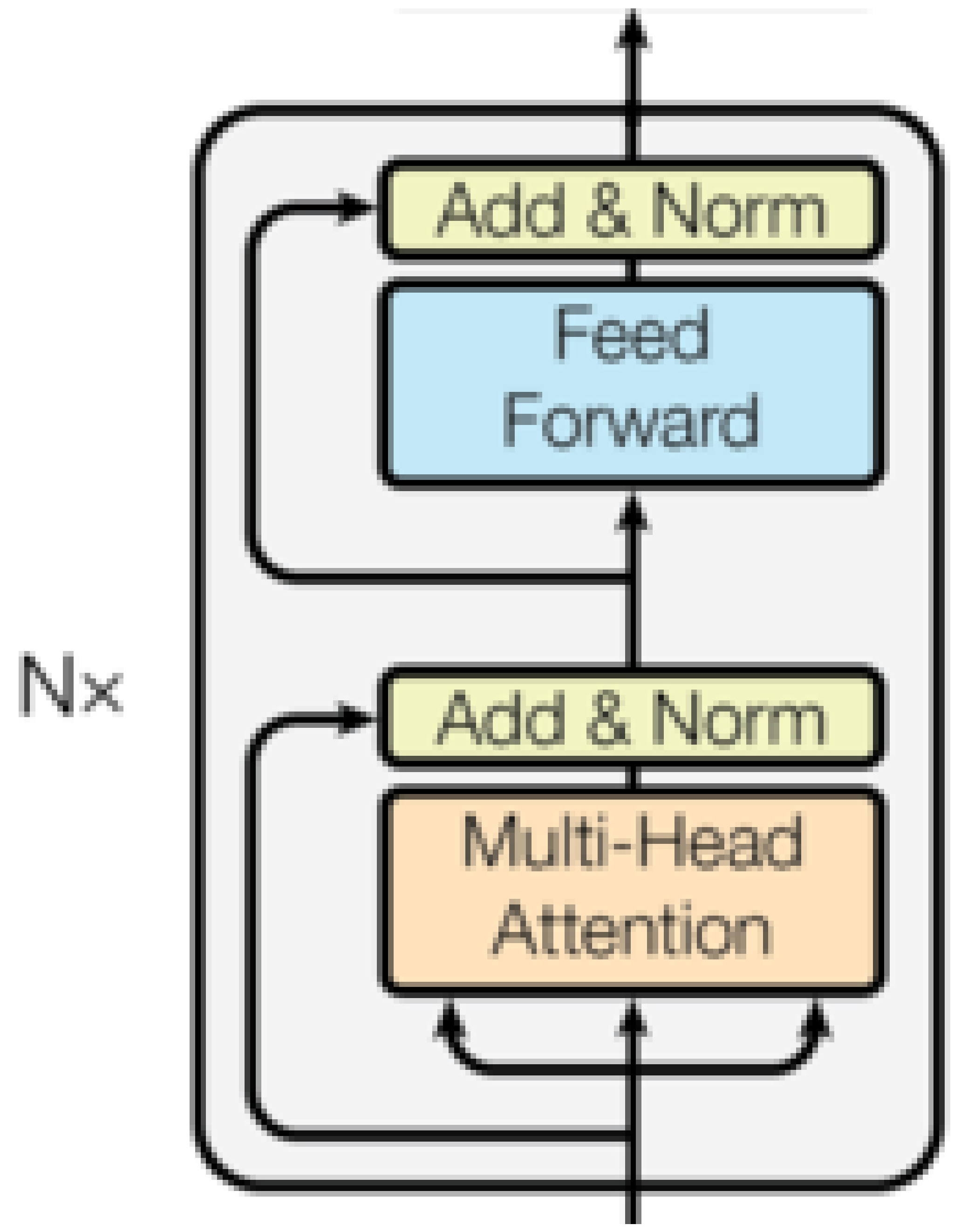
Multi head Self-Attention



Advantages of self-attention network

- Process in parallel
- Better in modeling long term dependencies, able to freely attend to other position.

Encoder : Add & Norm



- Residual Connection
- Layer Normalization

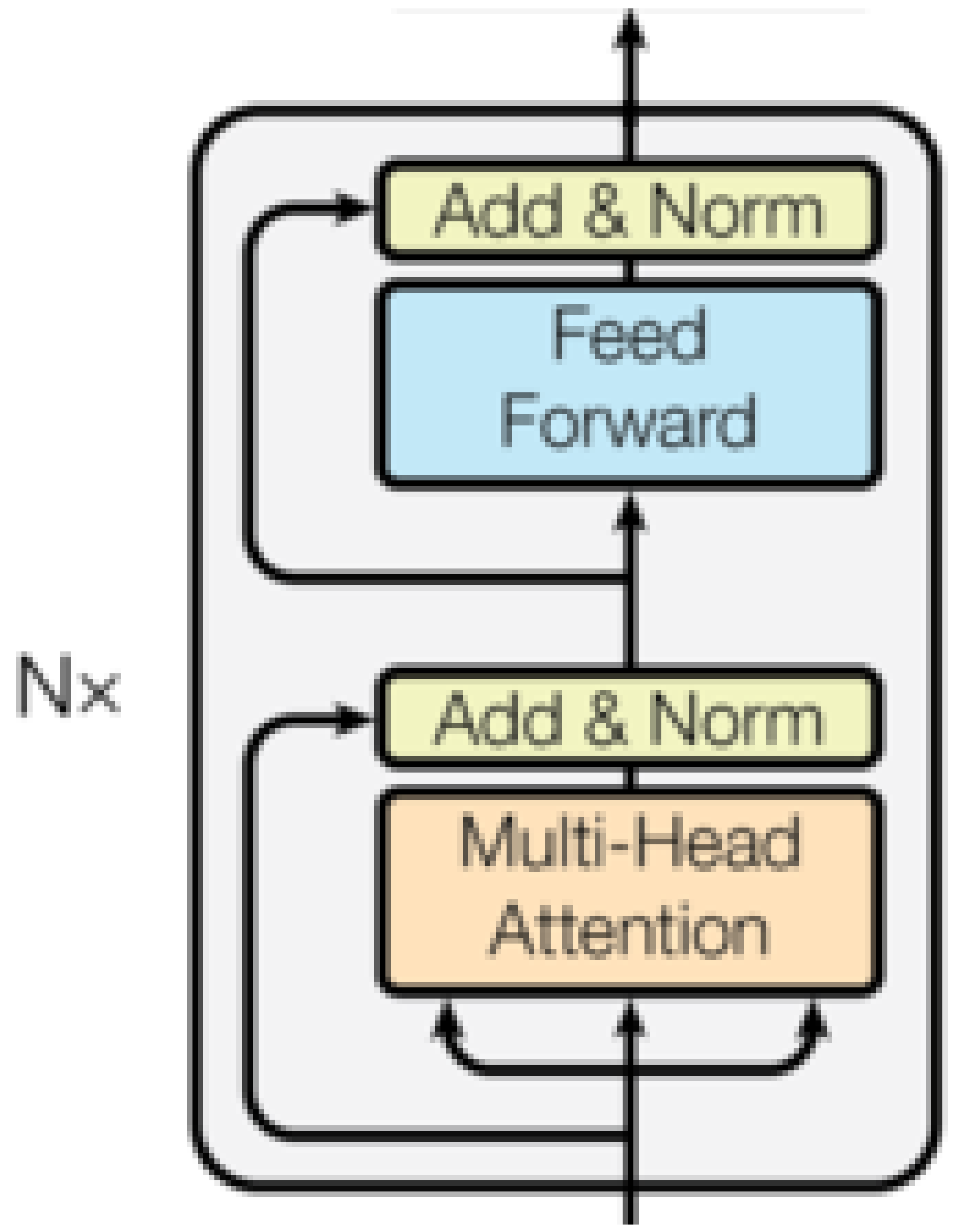
$$x' = \text{LayerNorm}\left(\text{SelfATT}(x) + x\right)$$

Layer Normalization – to normalize mean and variance of inputs (a^l) for a specific layer (l), assume that the layer has H hidden units.

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l \qquad \sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^l - \mu^l\right)^2} \qquad \bar{a}_i^l = \frac{g_i^l}{\sigma_i^l}\left(a_i^l - \mu_i^l\right)$$

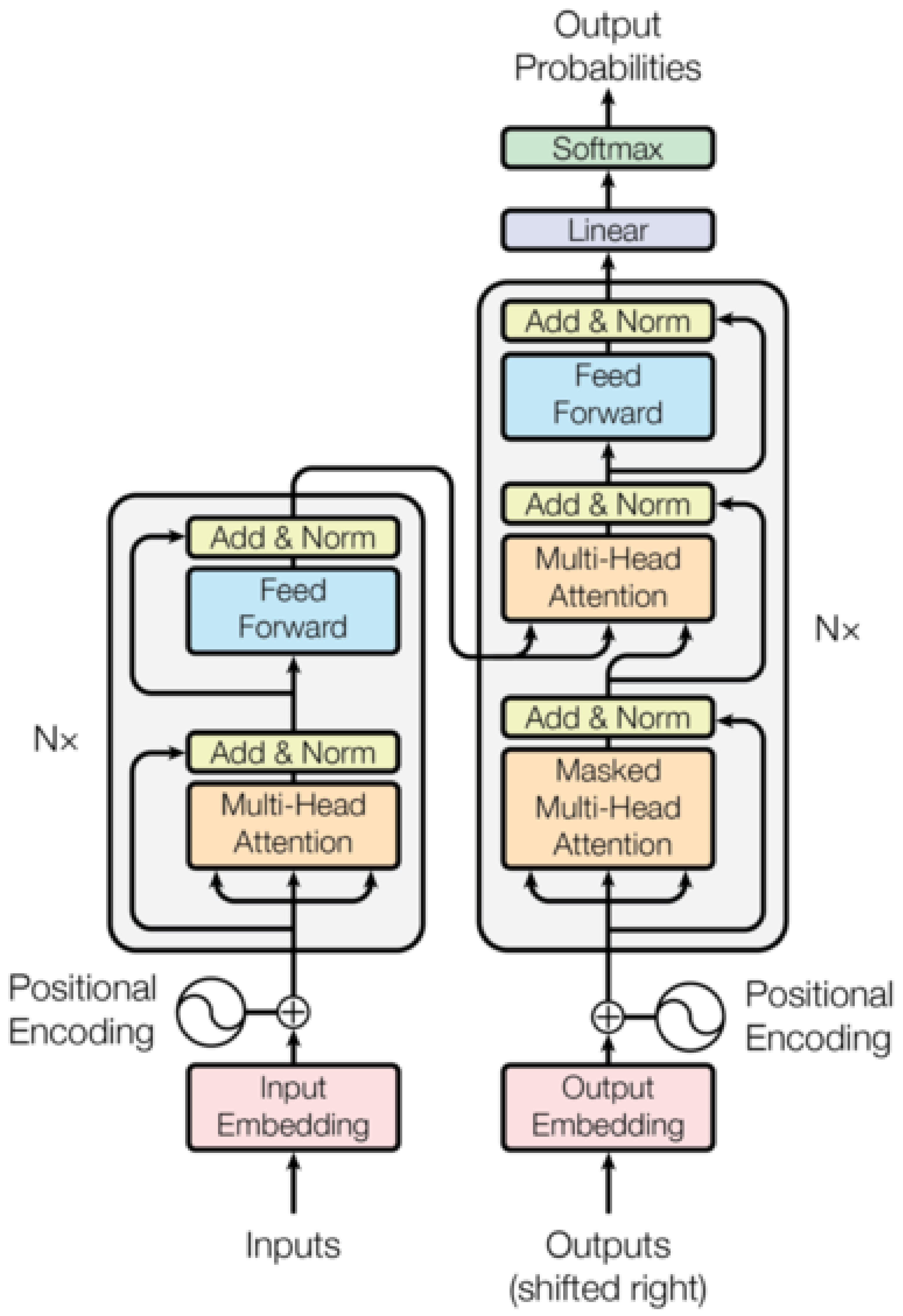
Encoder : Feed Forward

- Feed forward network
- Followed by residual connection and layer normalization



Transformer : Decoder

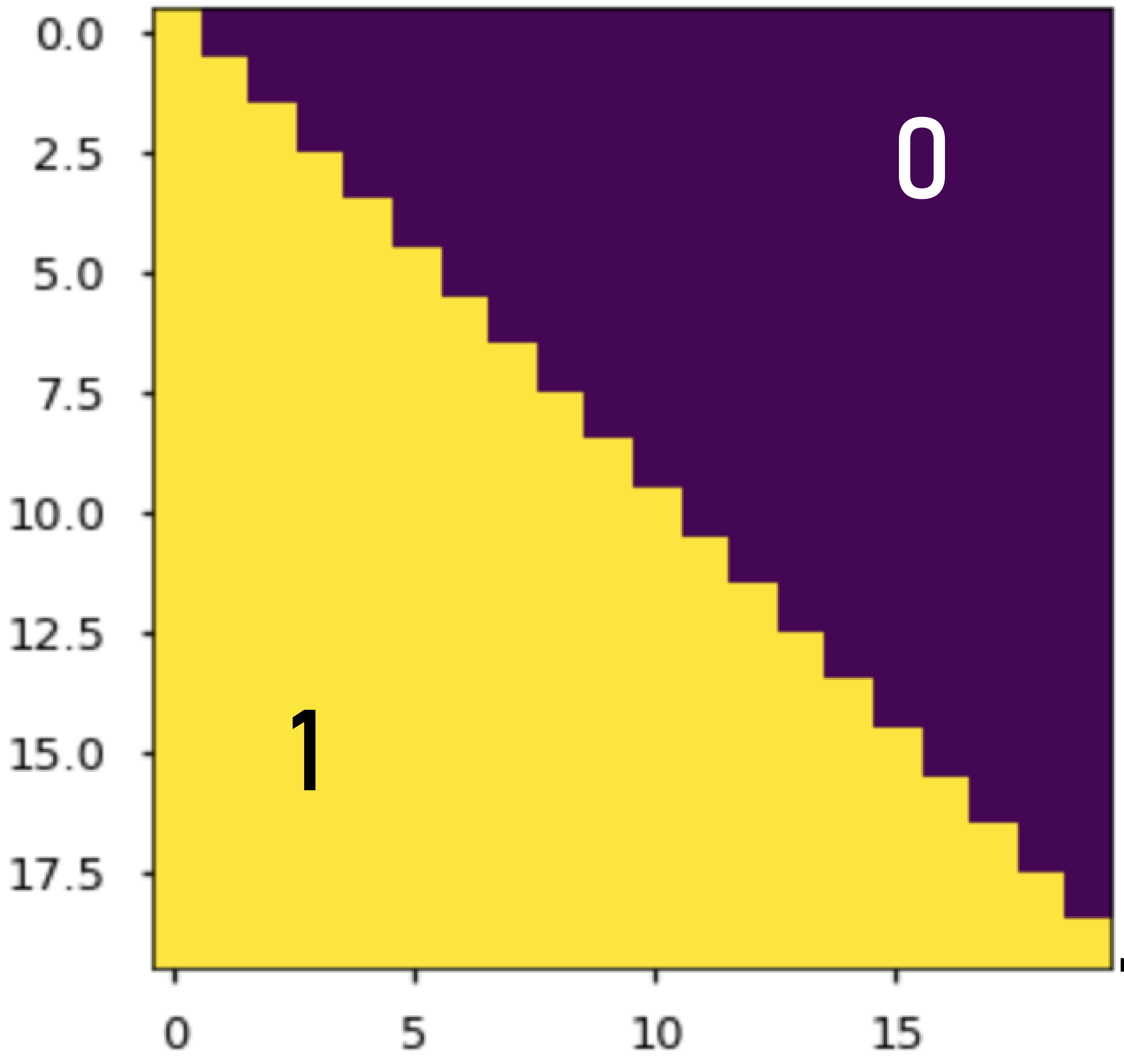
- Auto Regressive Decoding
- Based on Transformer layers



Decoder : Masked Multi-head Attention

- Auto Regressive Decoding

Current token can attend only left-side tokens because in decoding step the right-side tokens are not generated. Attention weights are multiplied by this mask matrix.

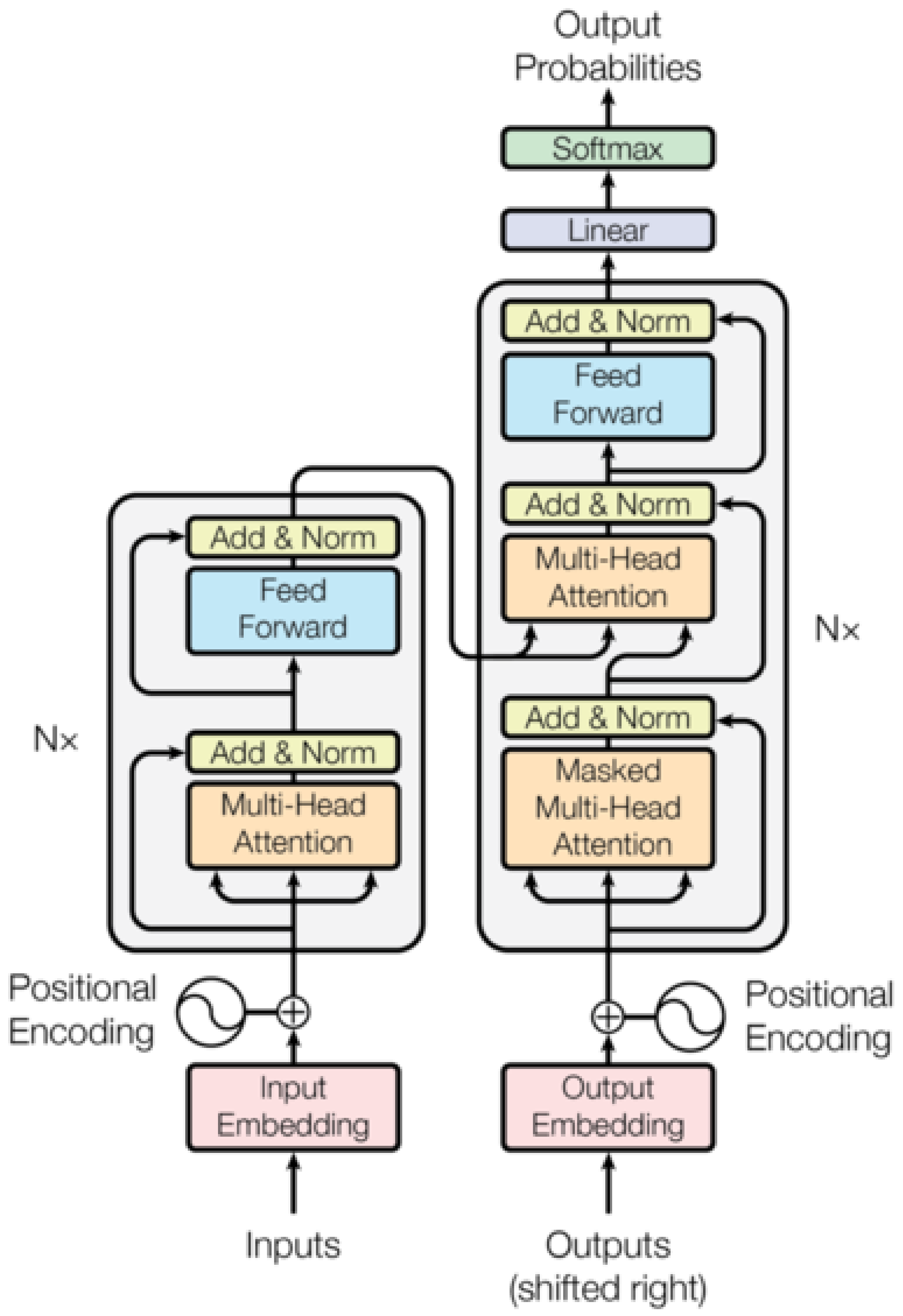


Mask vector for the first token. [1, 0, 0, 0, …, 0]

Mask vector for the last token. [1, 1, 1, 1, …, 1]

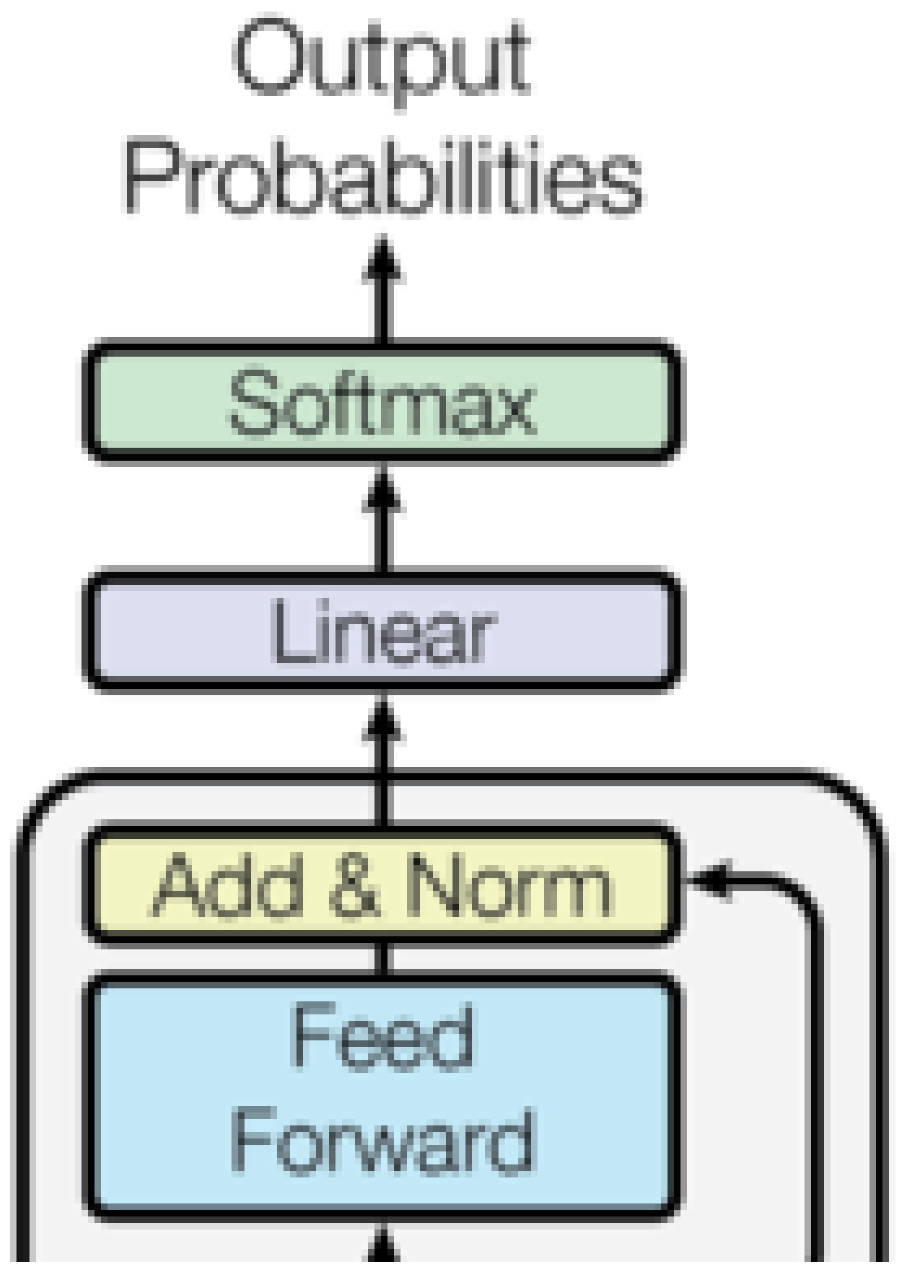
Decoder-Encoder Attention

- Query data from the encoder outputs.
- Encoder output (K,V) , Decoder State (Q)



Decoder : Feed Forward and Softmax

- Predict target word distribution



Optimization

- Label Smoothed Regularization

$$\bar{y}_j^t = (1-\epsilon)\bar{y}_j + \frac{\epsilon}{V}$$

For example, V = 3, $\epsilon = 0.3$

$$\bar{y}_{\text{true(smooth)}} = \begin{bmatrix}0\\0\\1\end{bmatrix} - \begin{bmatrix}0\\0\\\epsilon\end{bmatrix} + \frac{1}{3}\begin{bmatrix}\epsilon\\\epsilon\\\epsilon\end{bmatrix} = \begin{bmatrix}0.1\\0.1\\0.8\end{bmatrix}$$

Optimization

- Label Smoothed Regularization

$$\bar{y}_j^t = (1-\epsilon)\bar{y}_j + \frac{\epsilon}{V}$$

For example, $V = 3$, $\epsilon = 0.3$

$$\bar{y}_{\text{true(smooth)}} = \begin{bmatrix}0\\0\\1\end{bmatrix} - \begin{bmatrix}0\\0\\\epsilon\end{bmatrix} + \frac{1}{3}\begin{bmatrix}\epsilon\\\epsilon\\\epsilon\end{bmatrix} = \begin{bmatrix}0.1\\0.1\\0.8\end{bmatrix}$$

- Label Smoothed NLL

$$= -\sum_{t=1}^{T}\sum_{j=1}^{V}\bar{y}_j^t \log \hat{y}_j^t$$

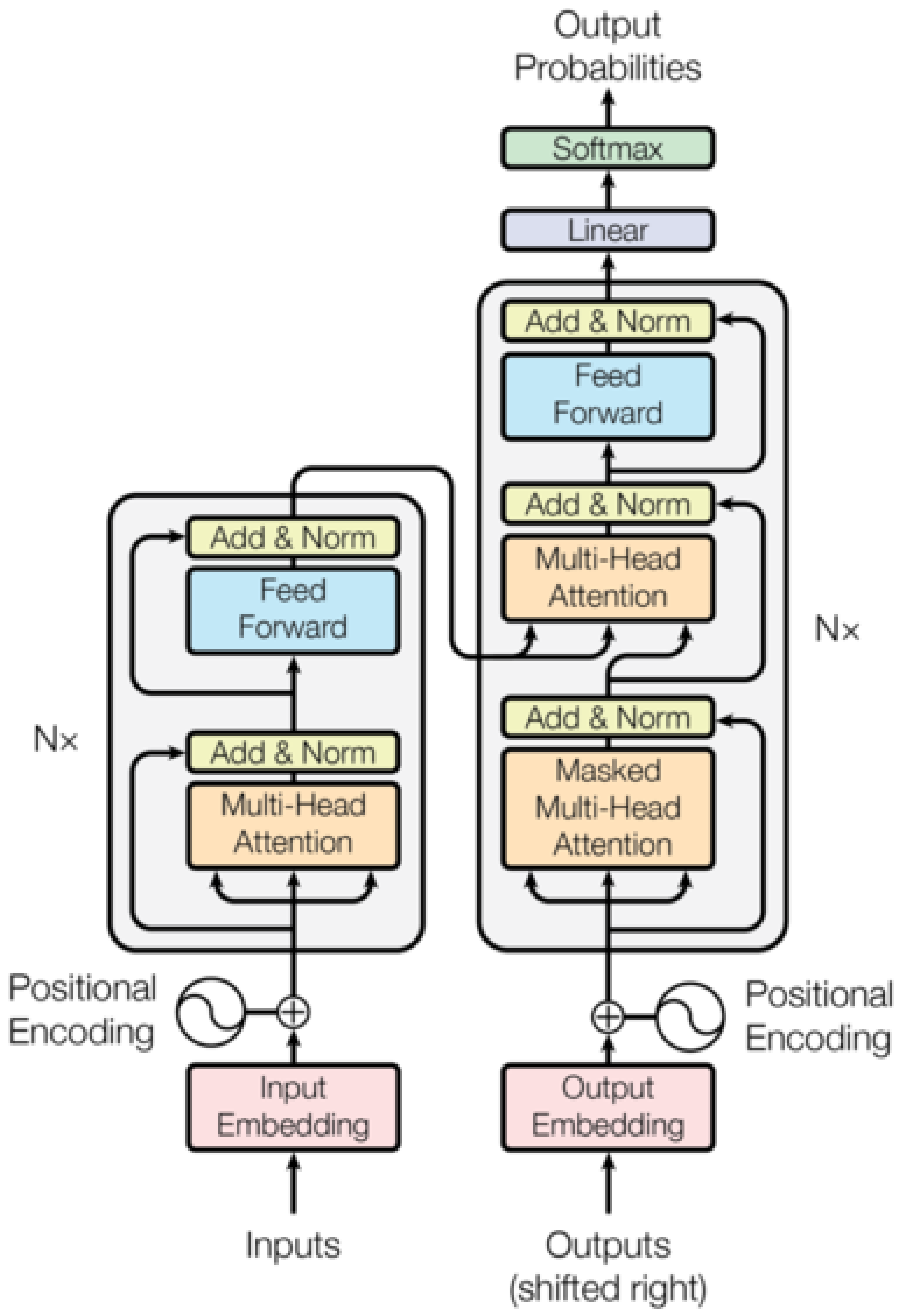
Decoding : Auto Regressive

- Greedy search
- Beam Search

https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html

Summary

- Self-Attention Network
- Multi-Head Self Attention
- Positional Encoding
- Encoder
- Decoder
- Optimization



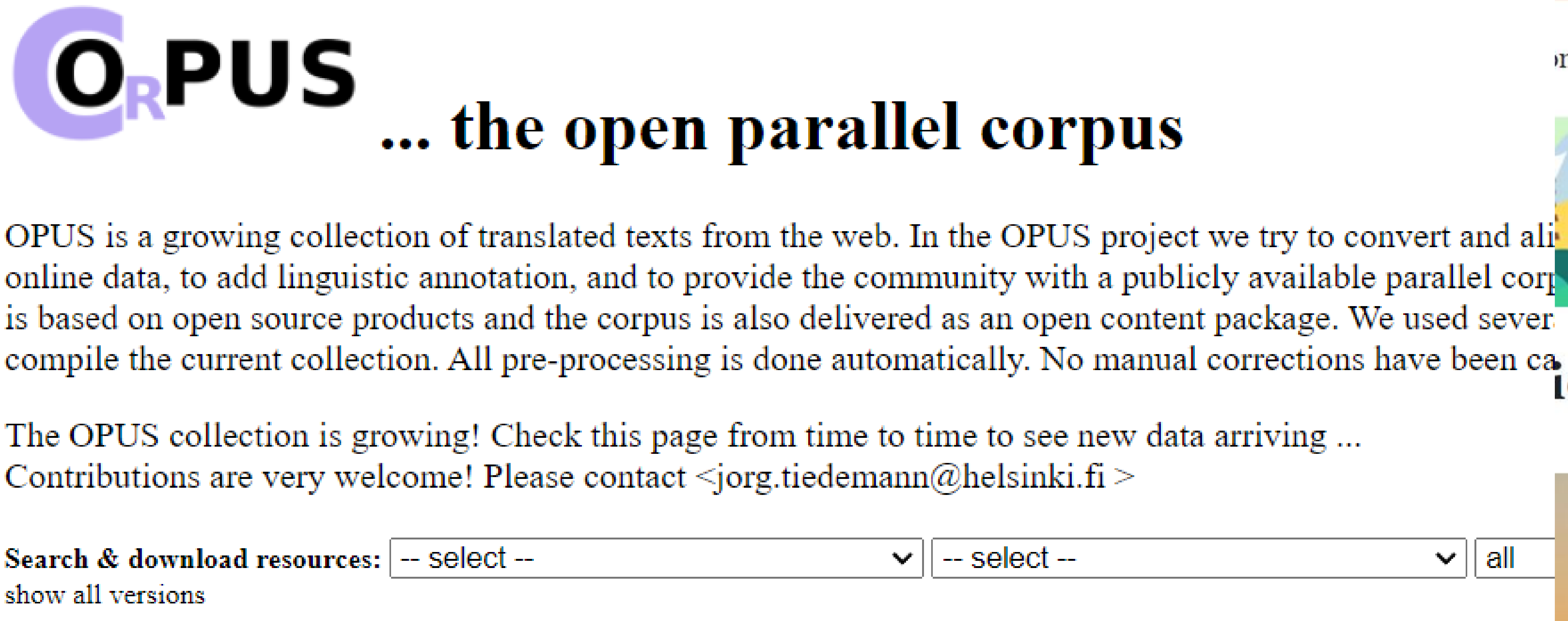
Data Preparation for MT

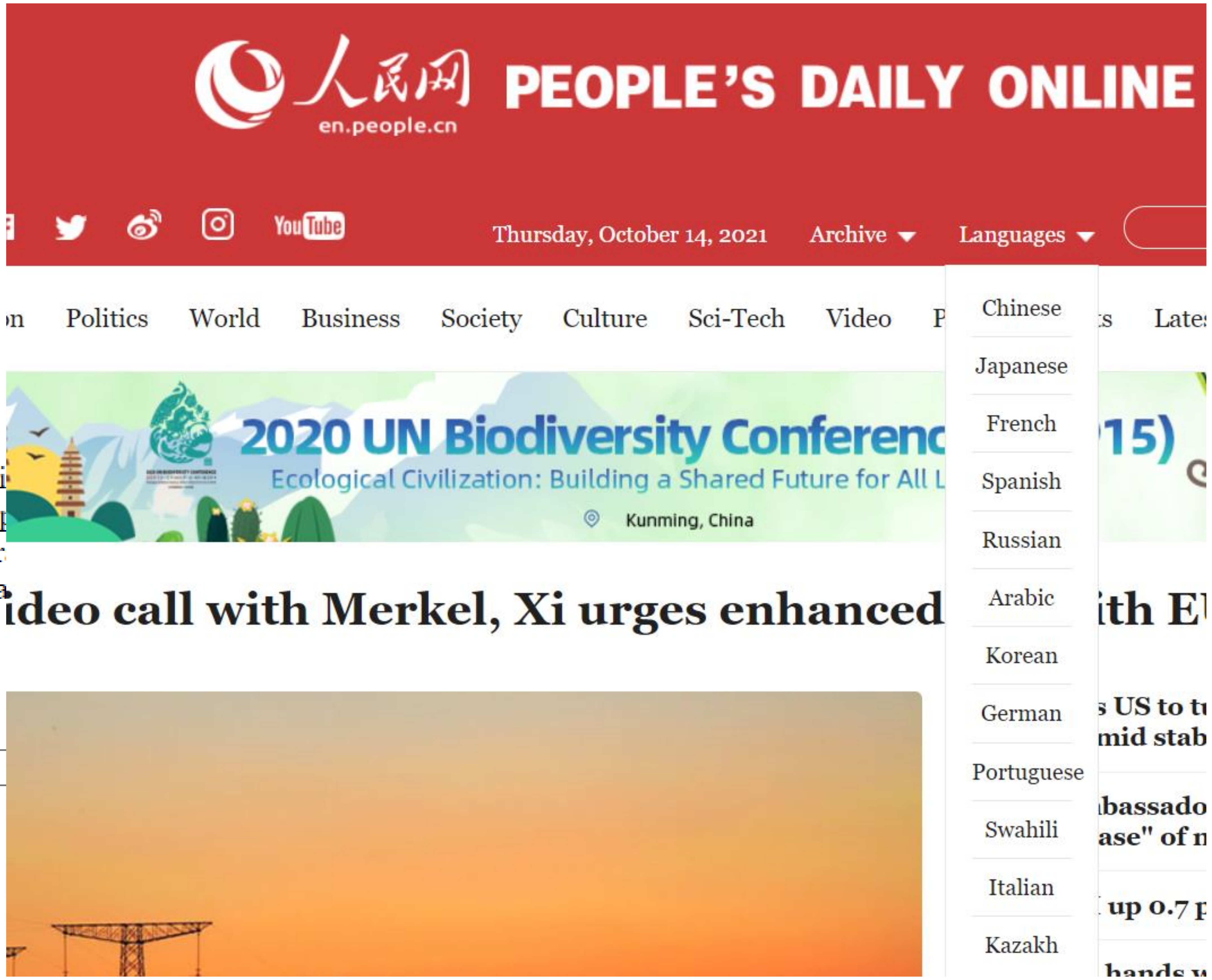
Data Preparation for MT

- **Data Collection**
- **Data Cleansing and Tokenization**
- **Split Train / Validation / Test**
- **Additional Preprocessing Steps for NMT**
  - Subword Preparation
  - Padding and Binarizing

Data Collection

- **Data Collection**



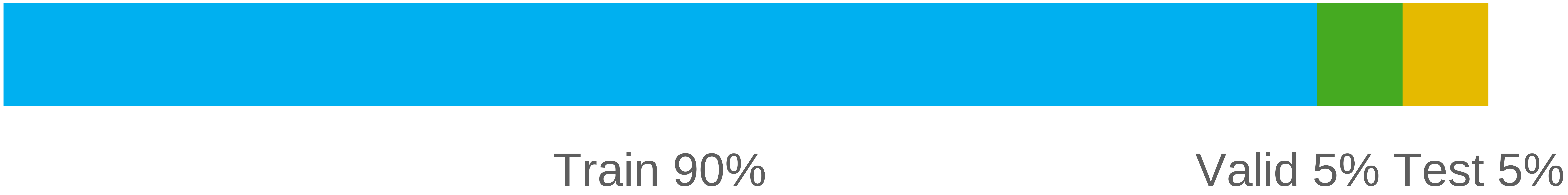


Data Cleansing and Tokenization

- Clean empty line
- Align parallel sentences
- Deduplicate
- Tokenize (Word, character segmentation)
- Filter low quality pairs
    - Alignment score
    - Length ratio (# of source tokens / # of target tokens)

Split Train / Validation / Test

- Shuffle
- Train / Validation / Test



Additional Steps for NMT

- Subword Units (Sentence piece, Byte-pair encoding)

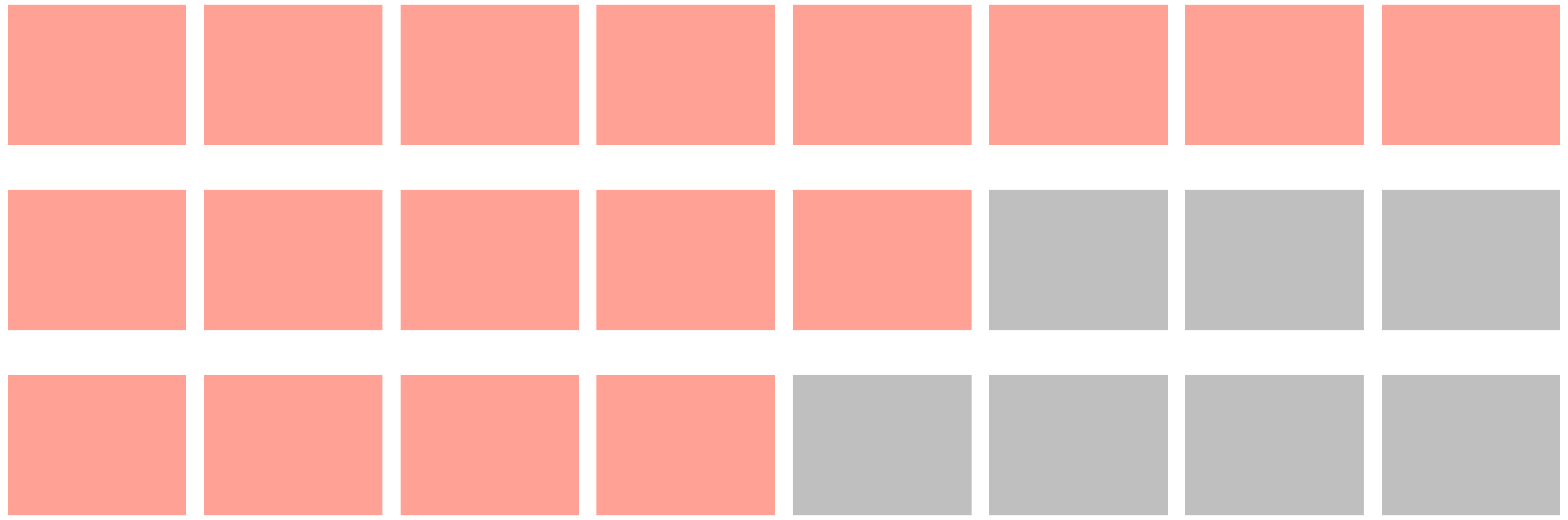
  ซีรีย์ จีน เรื่อง ดาบมังกรหยก

  ซีรีย์ จีน เรื่อง ดาบ@@ มังกร@@ หยก

  Rico Sennrich, Barry Haddow and Alexandra Birch, Neural Machine Translation of Rare Words with Subword Units, ACL, 2016

Additional Steps for NMT

- Padding



- Binarizing – Convert strings to tensors

Summary

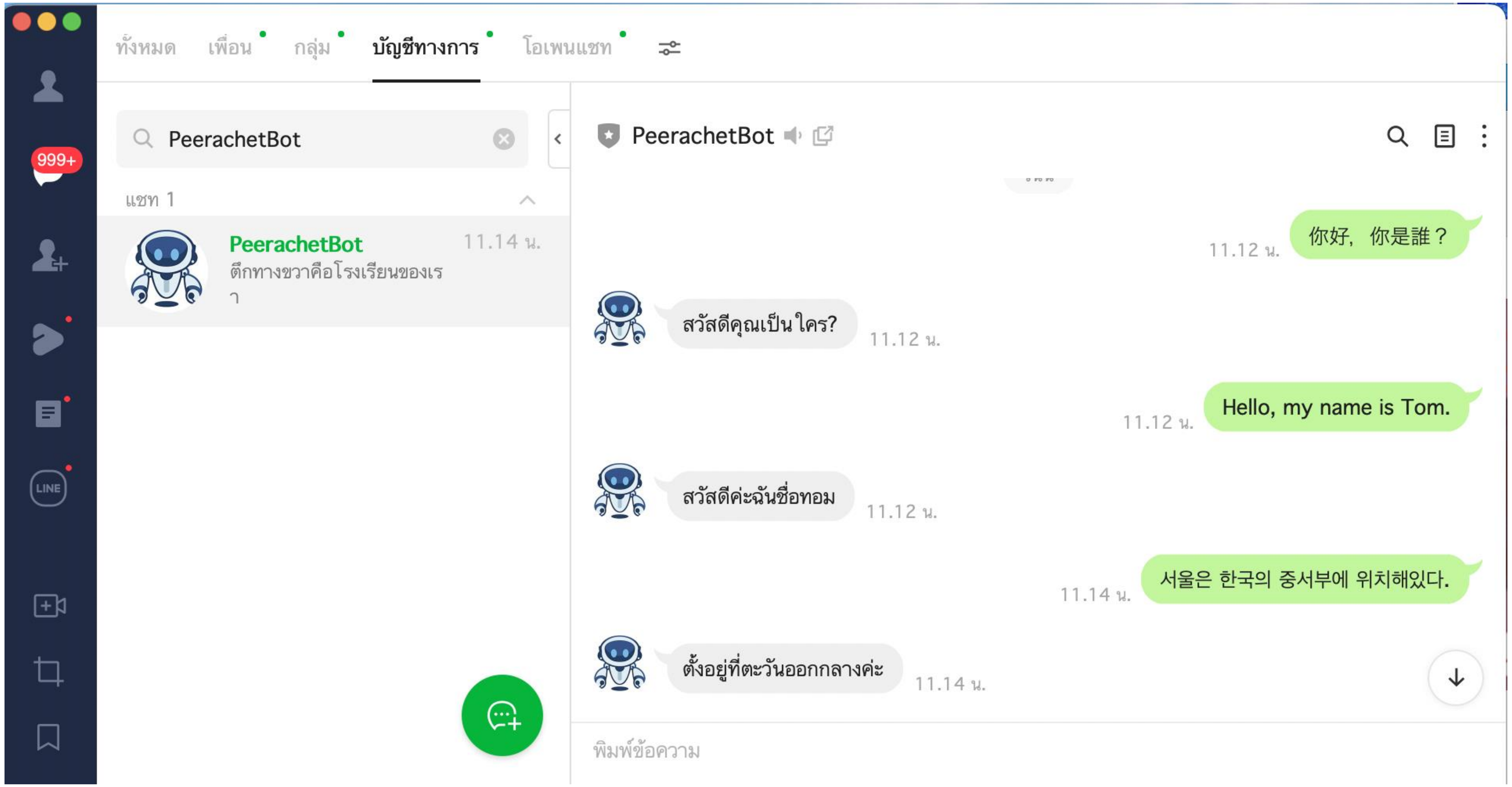
- **Data Collection**
- **Data Cleansing and Tokenization**
- **Split Train / Validation / Test**
- **Additional Preprocessing Steps for NMT**
    - Subword Preparation
    - Padding and Binarizing

Application of Seq2Seq Model

Possible Applications

- **Machine Translation**
- **Text Summarization**
- **Paraphrasing**
- **Question and Answering**
- **Chat Bot**

Line Bot Translation Service (Demo only)

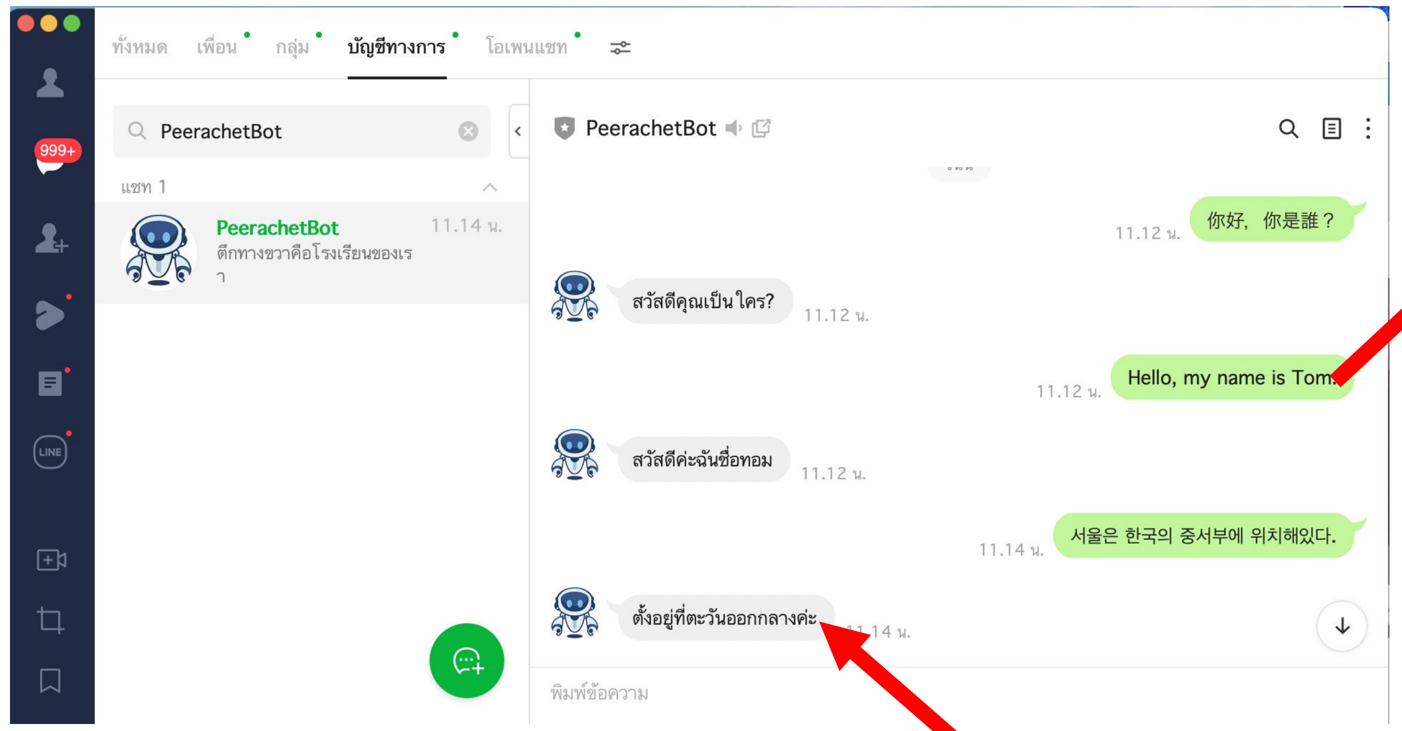


Bot information

Bot basic ID @198ldgap

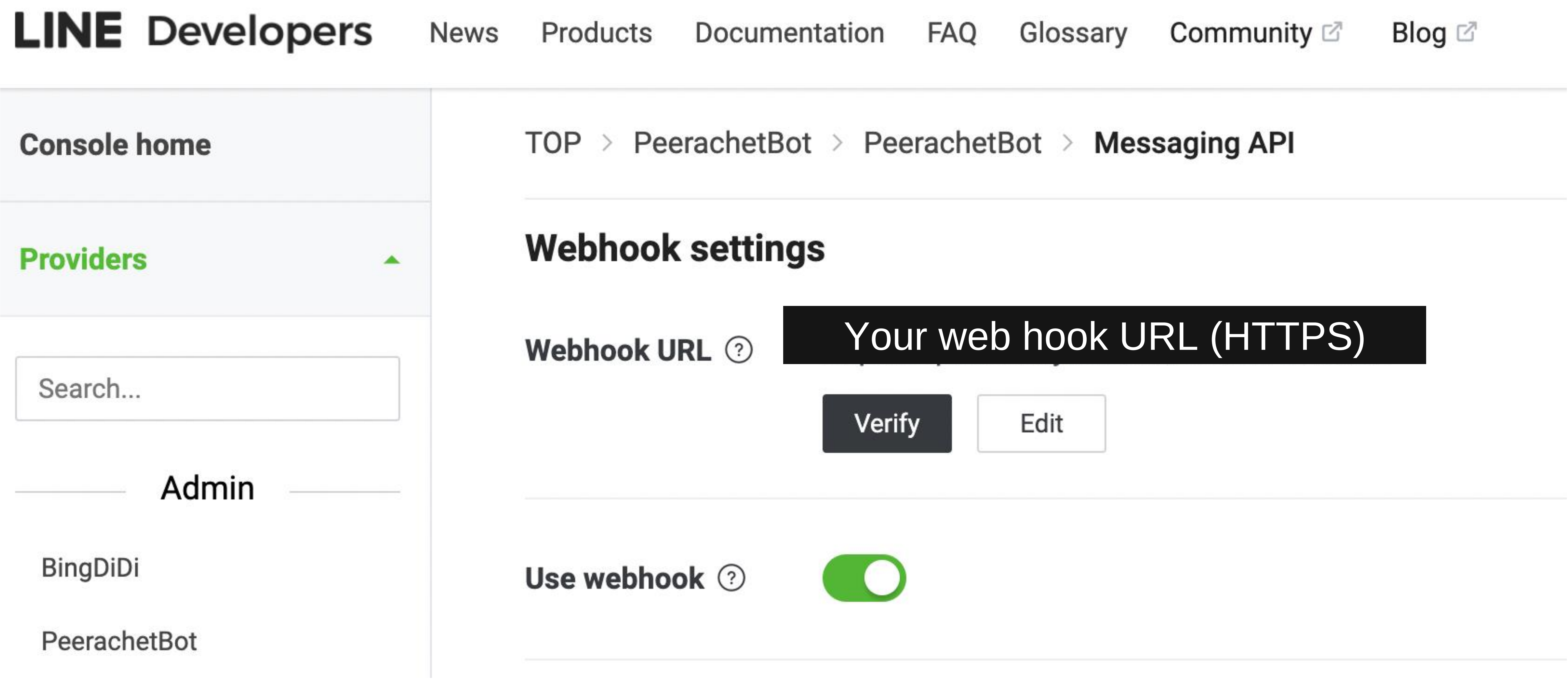
QR code

Line Bot Web Hook (Python version)




```python
@handler.add(MessageEvent, message=TextMessage)
def handle_text_message(event):
    text = event.message.text #message from user
    userid = str(event.source)

    print(json.loads(userid)["userId"])

    if detect(text) != "th":
        text = text[0:300]
        print("TRANSLATE ",text)
        now = datetime.now()
        # Format the date and time as a string
        date_time_str = now.strftime("%d/%m/%Y, %H:%M:%S")
        addlog(date_time_str,{"query" : text})


        response = requests.get(f"https://<your translation service url>/api.php?text={text}&lang=th")
        outtext = response.text
        line_bot_api.reply_message(
            event.reply_token,
            TextSendMessage(text=outtext)) #reply the same message from user

        return
```

LINE Developer Console



References

- StatMT.org
- https://archive.illc.uva.nl/ESSLLI2008/Materials/KoehnCallisonBurch/book.pdf
- Philipp Koehn, Statistical Machine Translation, 2009
- Christopher Olah, Understanding LSTM Networks, 2015
- Bahdanau, D., Cho, K. H., & Bengio, Y. Neural machine translation by jointly learning to align and translate. ICLR 20151
- Vaswani, Ashish & Shazeer, Noam & Parmar, Niki & Uszkoreit, Jakob & Jones, Llion & Gomez, Aidan & Kaiser, Luksz & Polosukhin, Illia, Attention is all you need , 2017
- Transformer: A Novel Neural Network Architecture for Language Understanding, 2017

References

- Jay Alammar, The Illustrated Transformer , 2018
- Rico Sennrich, Barry Haddow and Alexandra Birch, Neural Machine Translation of Rare Words with Subword Units, ACL, 2016
- บทที่1 ทำ LINE Bot สามารถโต้ตอบ หรือ Chatbot ด้วย Python (Official) - Saixiii

Thank you