Introduction to Transformer and GANs for NLP

2/2565: FRA501 Introduction to Natural Language Processing with Deep learning

Week 09

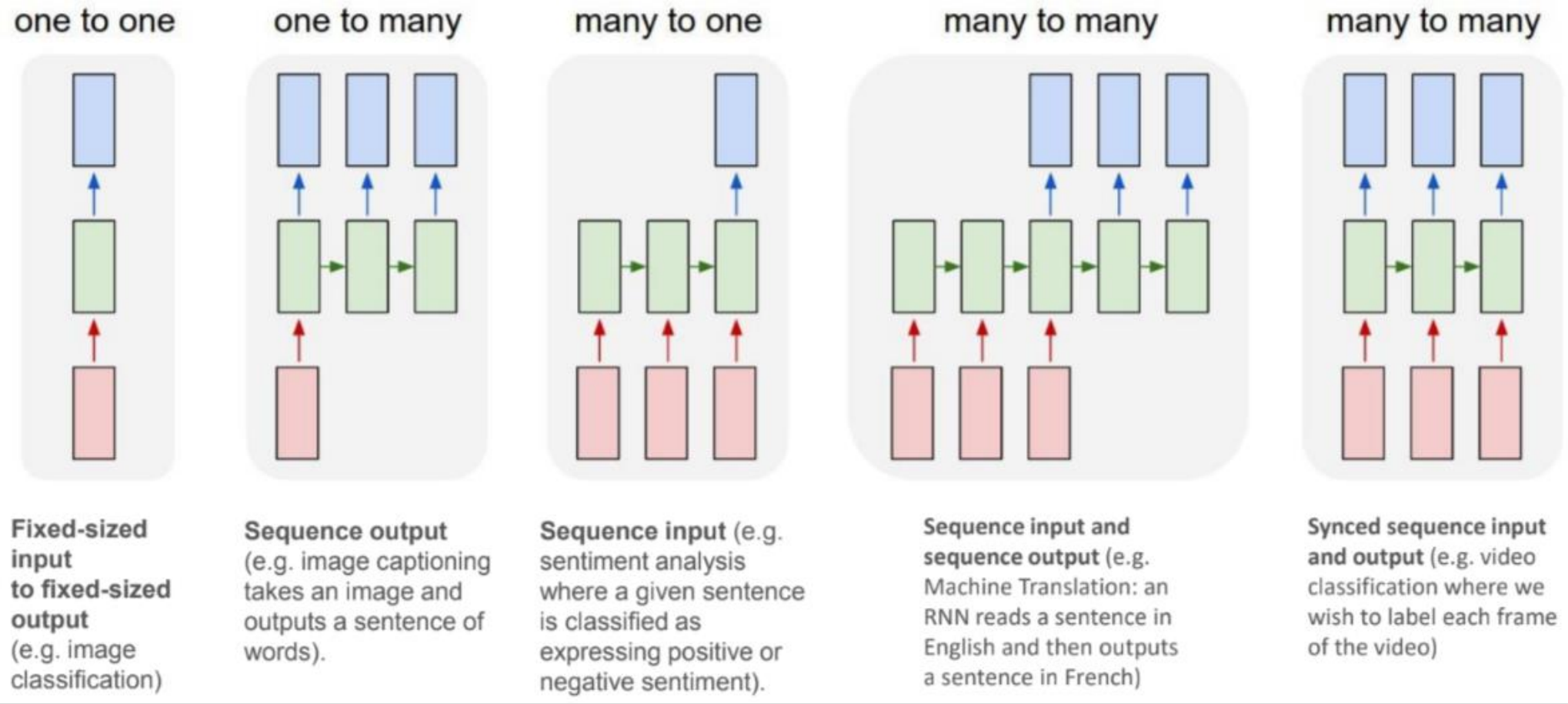
Paisit Khanarsa, Ph.D.

Institute of Field Robotics (FIBO), King Mongkut's University of Technology Thonburi

Outlines

- Recap of Attention Mechanism
- Transformer
- Beam Search
- GAN for NLP

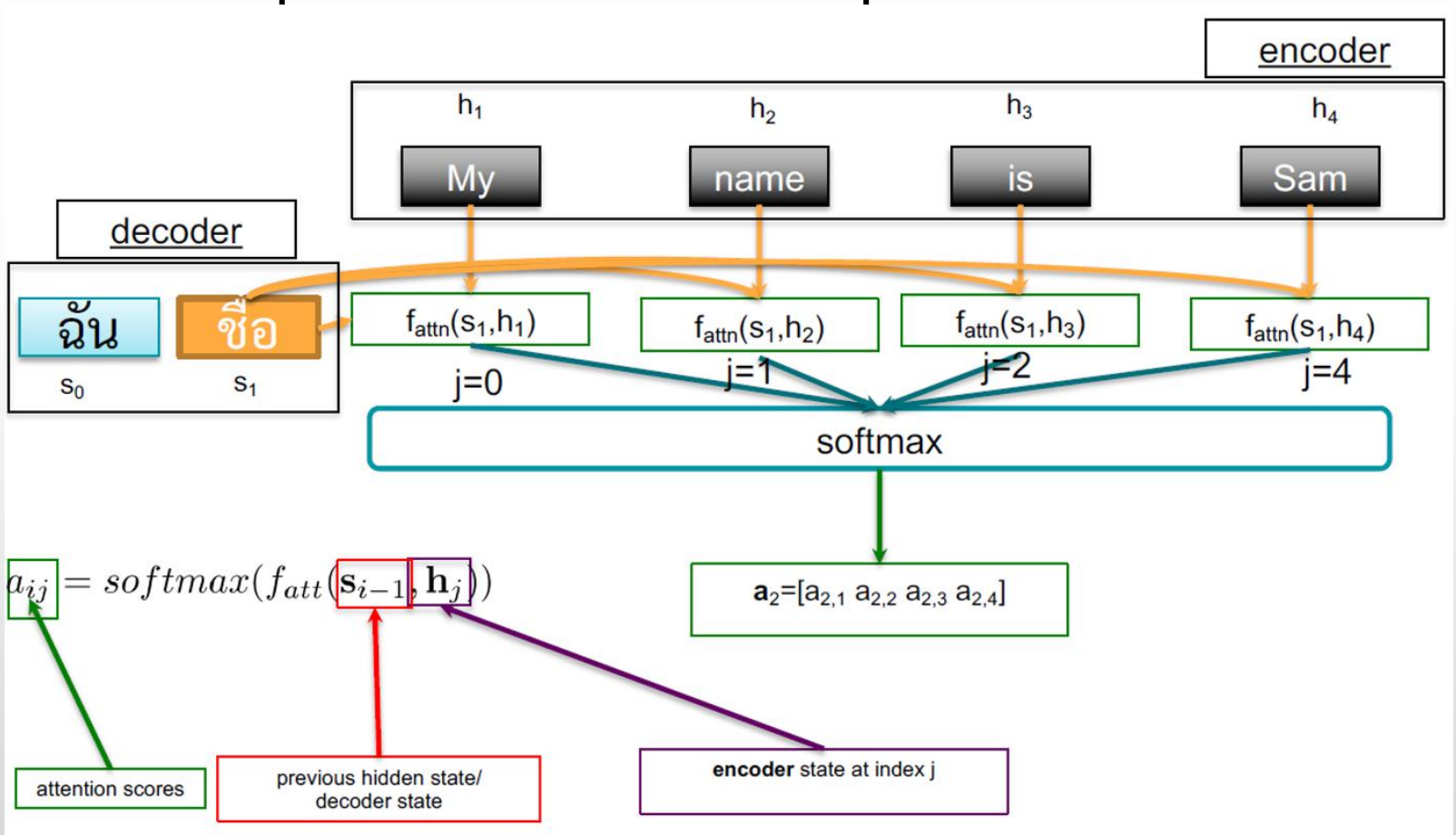
Recap of Attention Mechanism

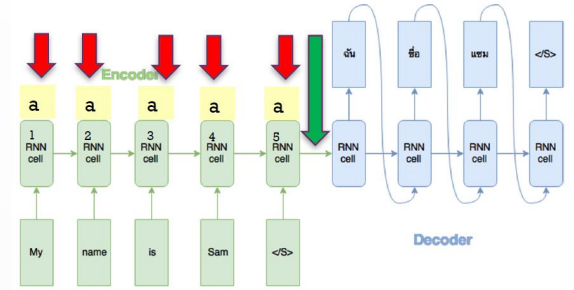


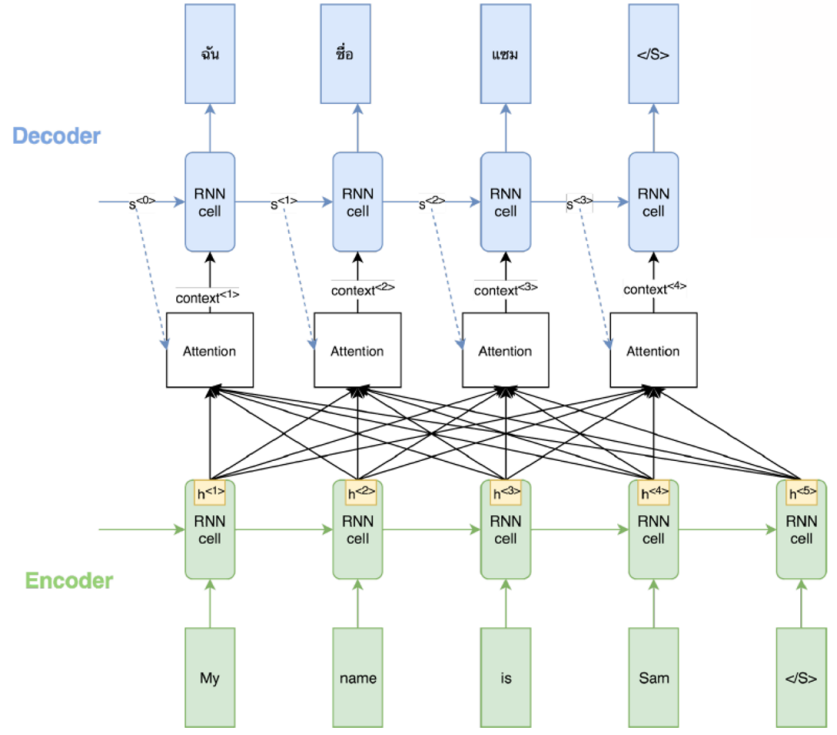
i:decoder index
j: encoder index

Attention Calculation: Attention Scores

- Example: Now we want to predict “ชื่อ”



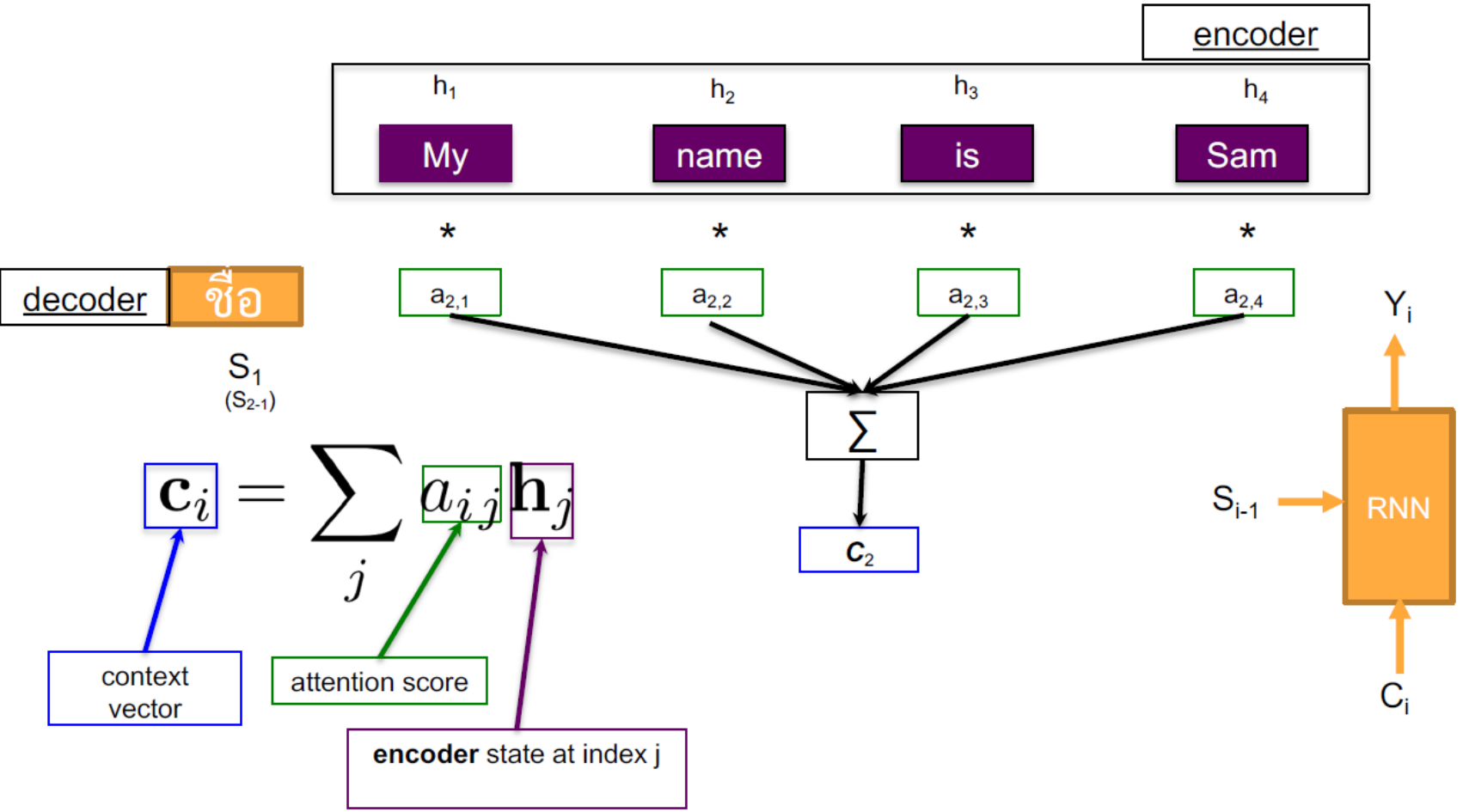


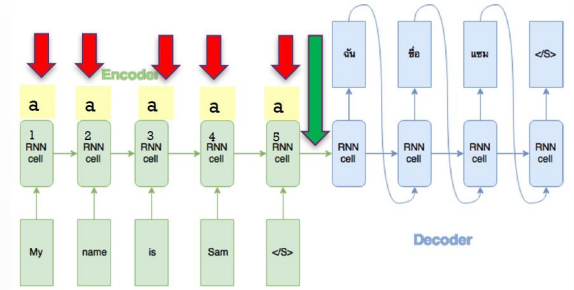


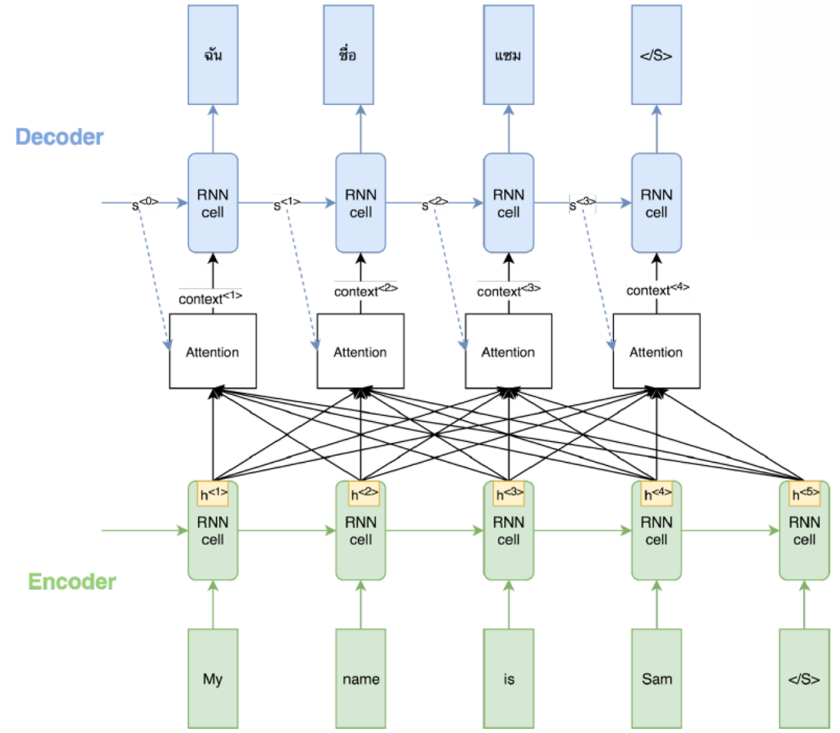
Attention Calculation: Context Vector

i:decoder index
j: encoder index

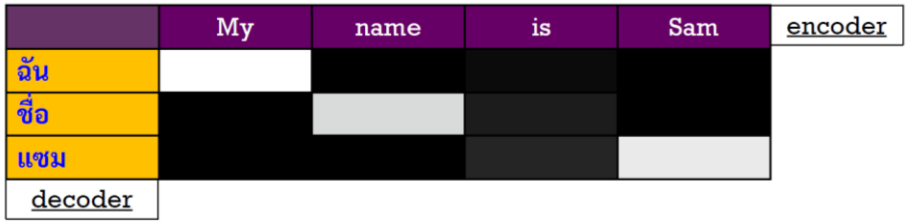
- Example: Now we want to predict “ชื่อ”







Type of Attention Mechanisms



$$a_{ij} = softmax(f_{att}(s_{i-1}, h_j))$$

- **Additive attention:** The original attention mechanism (Bahdanau et al., 2015) uses a one-hidden layer feed-forward network to calculate the attention alignment:

$$f_{att}(s_{i-1}, h_j) = tanh(\mathrm{W}_a[s_{i-1}; h_j])$$

- **Multiplicative attention:** Multiplicative attention (Luong et al., 2015) simplifies the attention operation by calculating the following function:

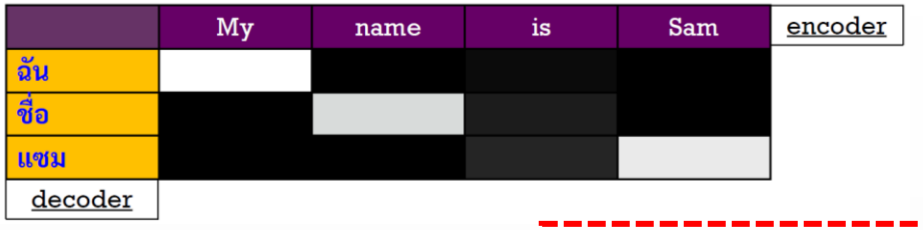
$$f_{att}(s_{i-1}, h_j) = \mathrm{S}_{i-1}^T W_a h_j$$

- **Self-attention:** Without any additional information, however, we can still extract relevant aspects from the sentence by allowing it to attend to itself using self-attention (Lin et al., 2017)

$$a = softmax(W_{s_2} \tanh(W_{s_1} H^T))$$

- **Key-value attention:** key-value attention (Daniluk et al., 2017) is a recent attention variant that separates form from function by keeping separate vectors for the attention calculation.

Self Attention



$$a_{ij} = softmax(f_{att}(s_{i-1}, h_j))$$

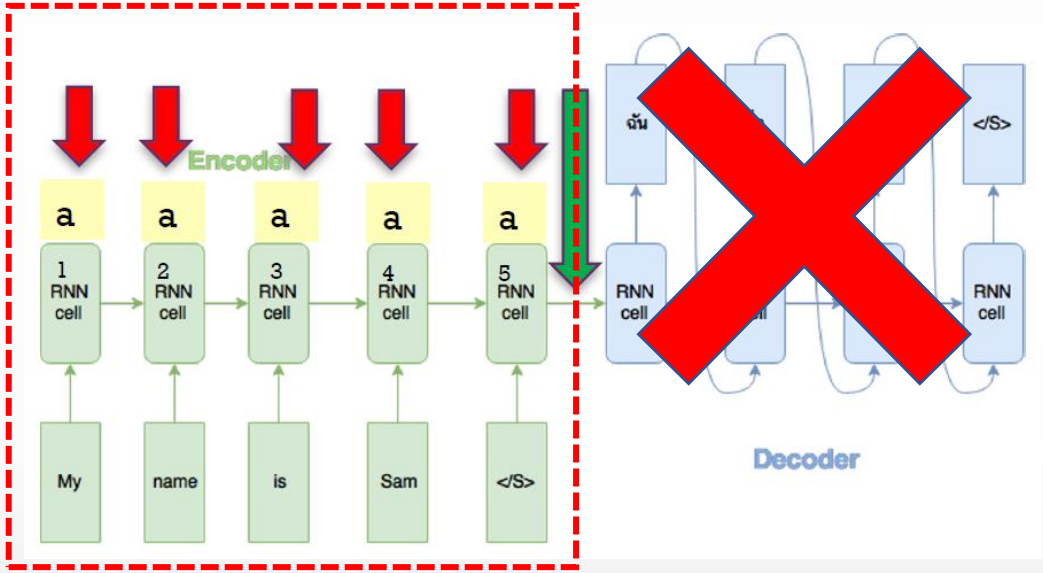
- Without any additional information, we can still extract relevant aspects from the sentence by allowing it to attend to itself using self-attention (Lin et al., 2017)

$$H = (\mathrm{h}_1, h_2, \ldots h_n)$$

Fully connected layer

$$a = softmax(W_{s_2} \tanh(W_{s_1} H^T))$$

One-hidden layer (Dense)



Additive Attention

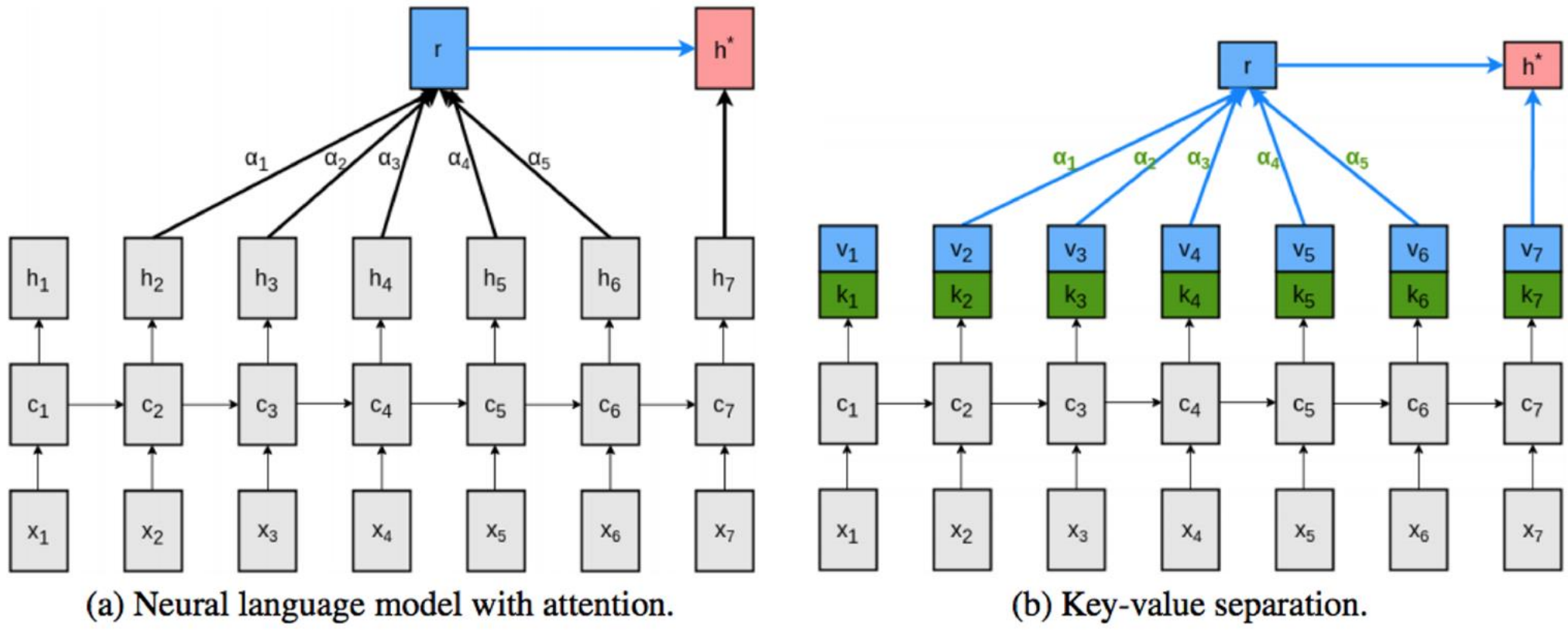
$$a_{ij} = softmax(tanh(\mathrm{W}_a[s_{i-1}; h_j]))$$

- w_{s_1} is a weight matrix, w_{s_2} is a vector of parameters. Note that these parameters are tuned by the neural networks.
- The objective is to improve a quality of embedding vector by adding context information.

Key-value attention

$$a_{ij} = softmax(f_{att}(s_{i-1}, h_j))$$

$$c_i = \sum_j a_{ij} h_j$$

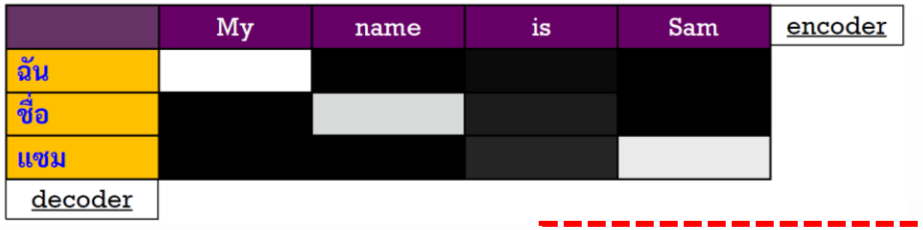


Value=encoded vector

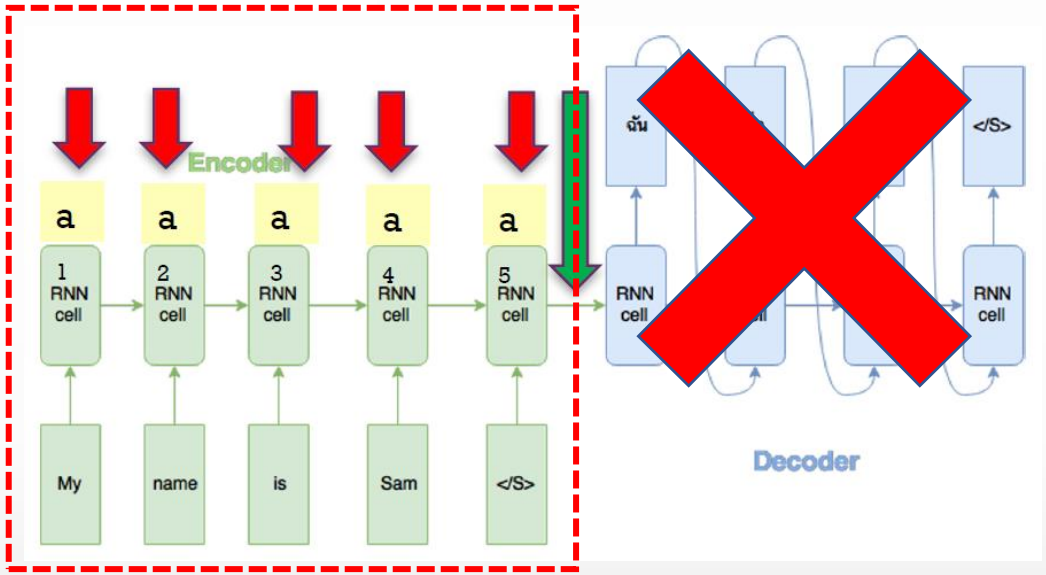
Key=used for attention score calculation

Daniluk, M., Rocktäschel, T., Welbl, J., & Riedel, S. (2017). Frustratingly short attention spans in neural language modeling. arXiv preprint arXiv:1702.04521.

Self Attention



$$a_{ij} = softmax(f_{att}(s_{i-1}, h_j))$$



- Without any additional information, we can still extract relevant aspects from the sentence by allowing it to attend to itself using self-attention (Lin et al., 2017)

$$H = (\mathrm{h}_1, h_2, \ldots h_n)$$

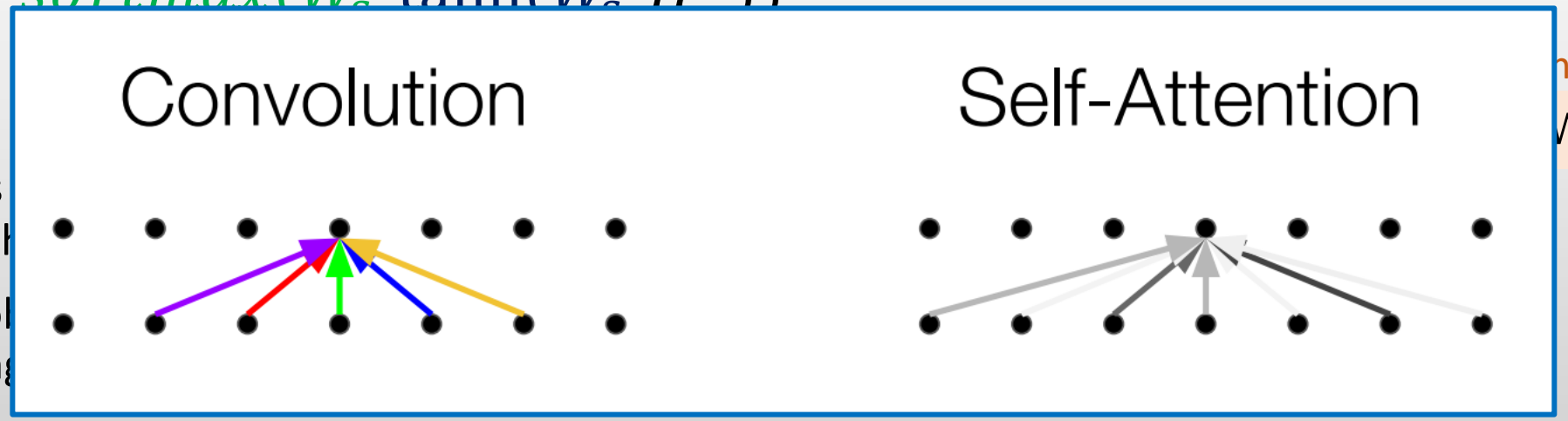
Fully connected layer

$$a = softmax(W_s \tanh(W_s H^T))$$

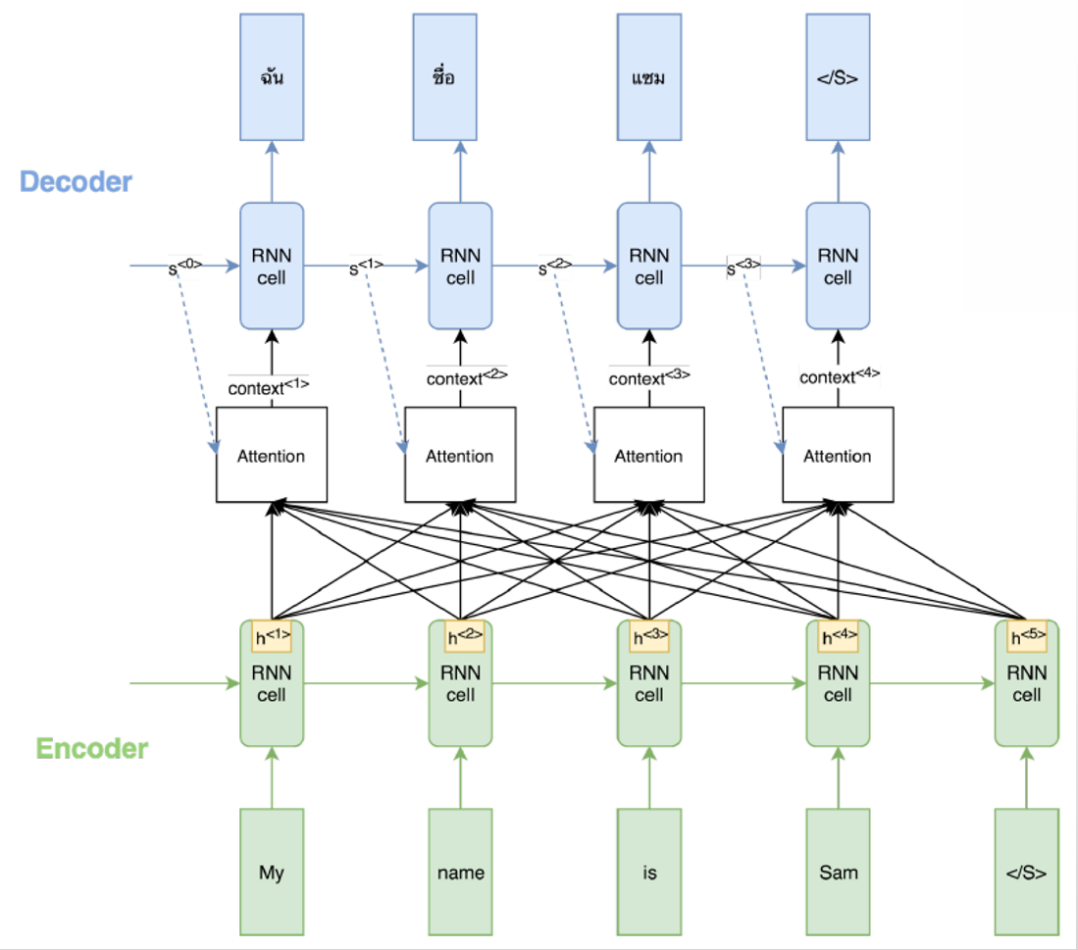
ntion

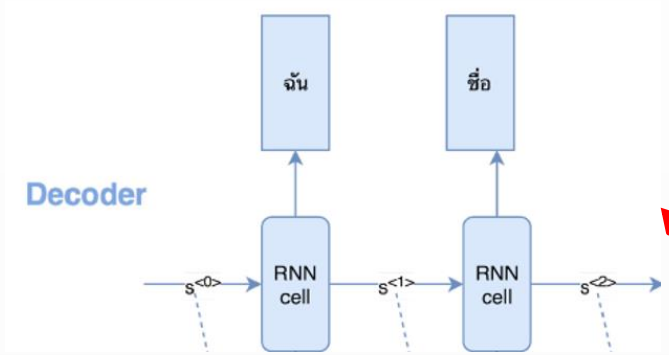
$W_a[s_{i-1}; h_j]))$

- w_{s_1} is
  that th
- The ol
  adding



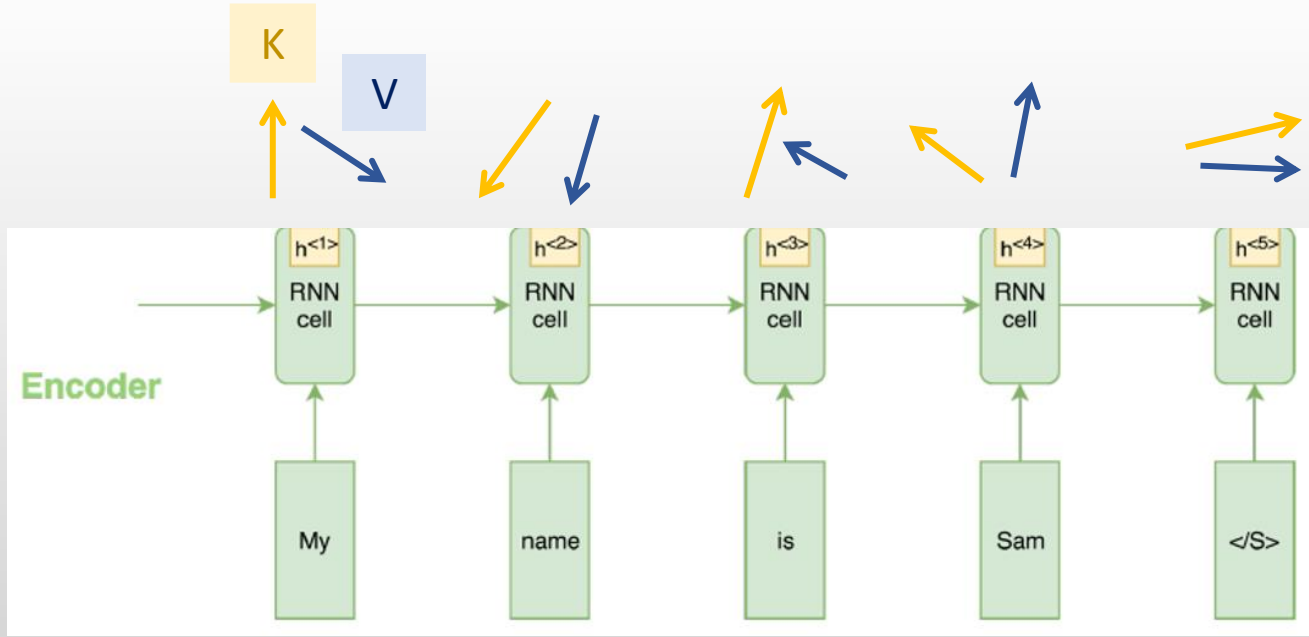
Query, Key and Value in Attention mechanism





$$a_{ij} = softmax(f_{att}(s_{i-1}, h_j))$$

$$c_i = \sum_j a_{ij} h_j$$



Transformer

- Attention is all you need

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*.

https://arxiv.org/pdf/1706.03762.pdf

Model Architecture of Transformer

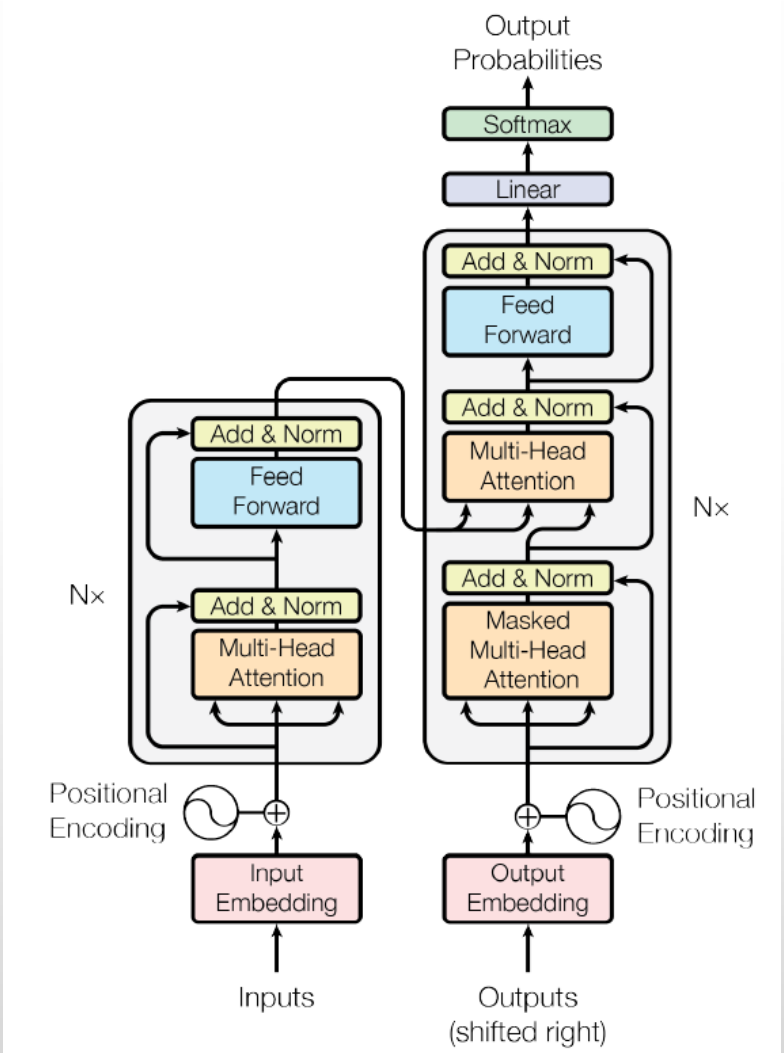


Figure 1: The Transformer - model architecture.

https://arxiv.org/pdf/1706.03762.pdf

Model Architecture of Transformer

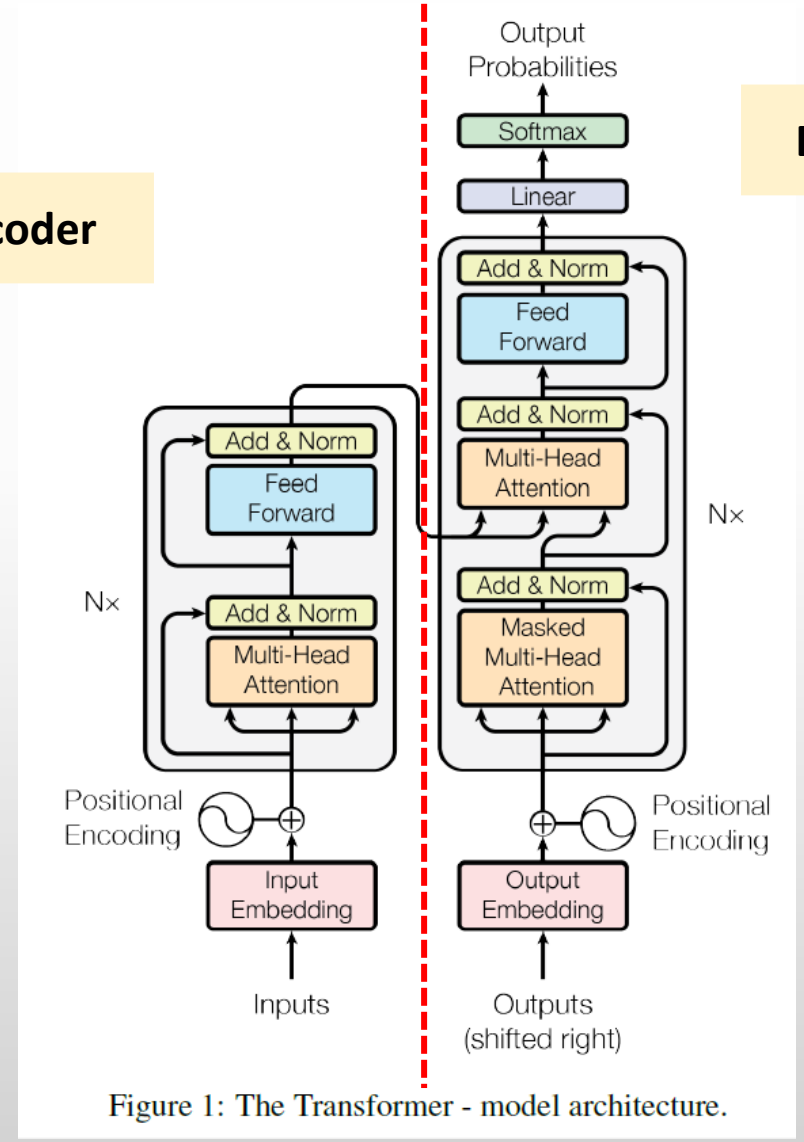


Figure 1: The Transformer - model architecture.

https://arxiv.org/pdf/1706.03762.pdf

Model Architecture of Transformer

Example:

The black cat is on the black chair

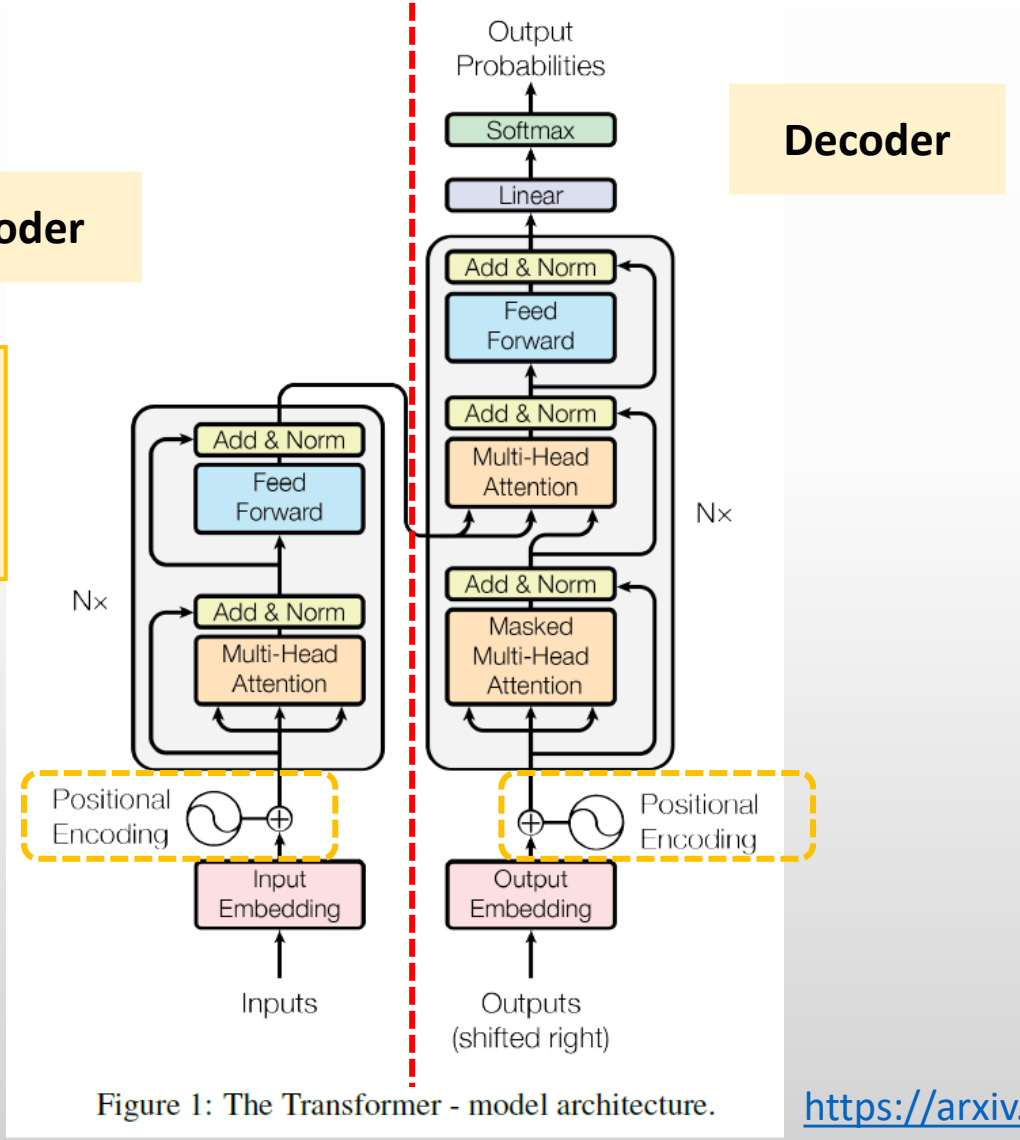


Figure 1: The Transformer - model architecture.

https://arxiv.org/pdf/1706.03762.pdf

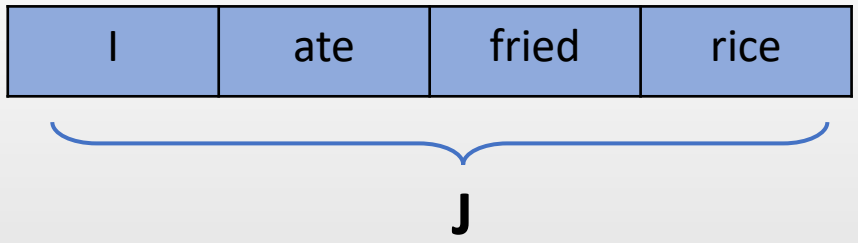
Recap: Positional Encoding

j=positional index
d=dimensional index

- Positional Encoding (PE)
  - Position is modeled by a multiplicative term on each word vector with weights depending on the position in the sentence

$$f(j,d) = (1 - j/J) - (d/D)(1 - 2j/J)$$

S =



PE(I) =

I			
1	**0.13**	**O**	**0.70**
2	0.04	O	0.65
…	…	O	…
…	…	O	…
D-1	0.91	O	0.30
D	0.82	O	0.25

=

0.09
0.03
…
…
0.27
0.20

O is element-wise multiplication

Model Architecture of Transformer

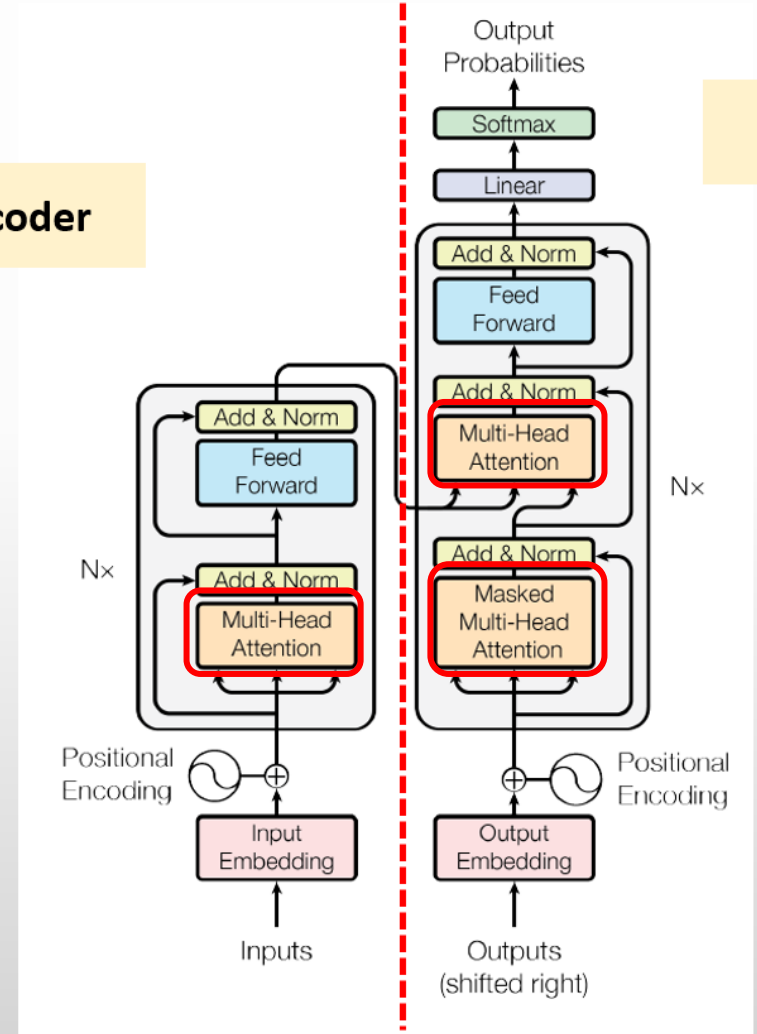
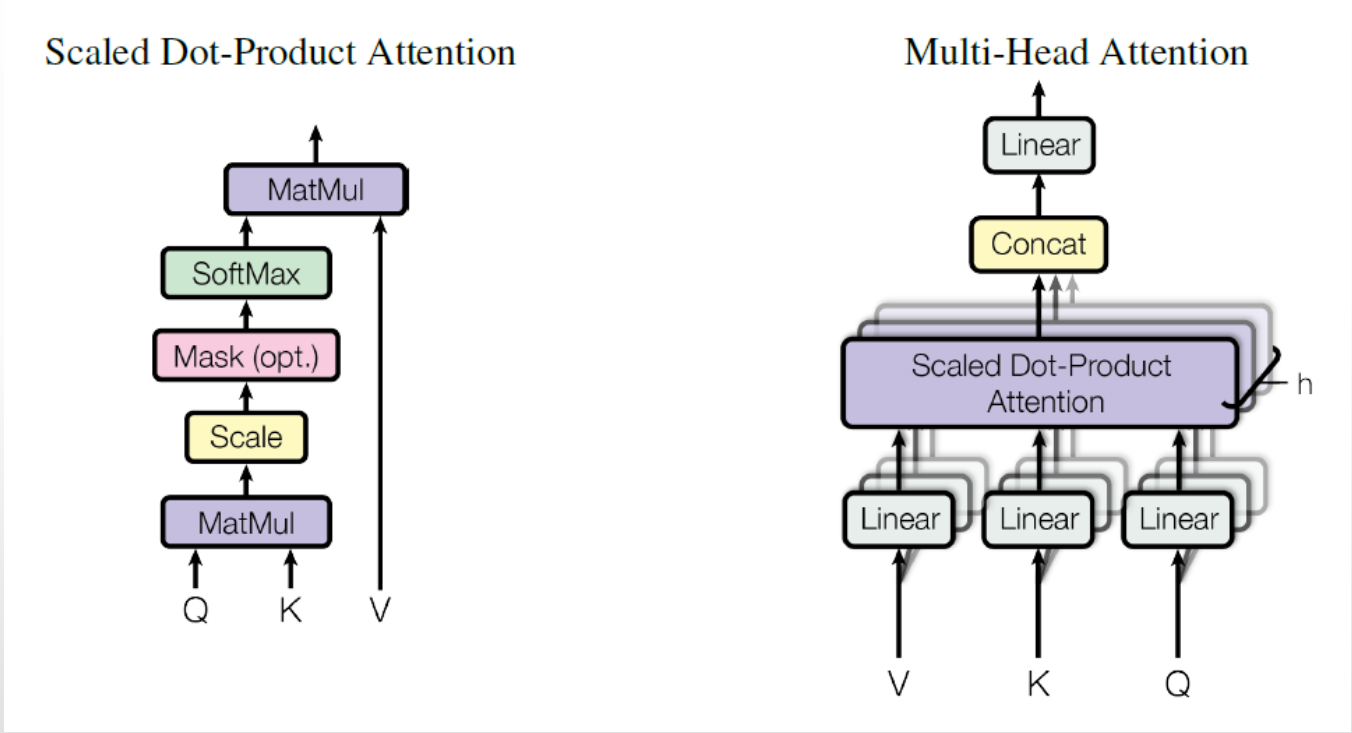
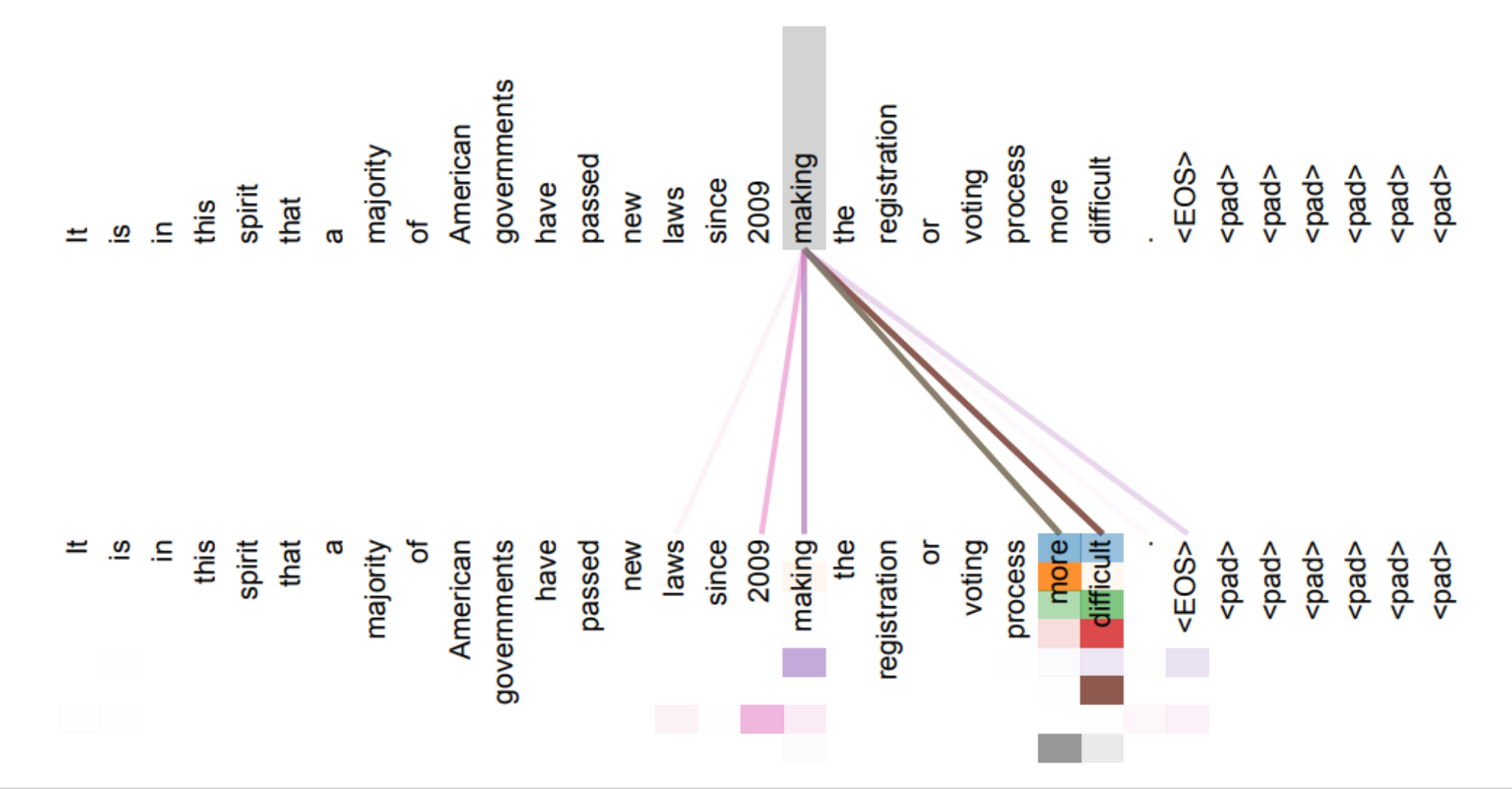


Figure 1: The Transformer - model architecture.



- Query – used with Key to determine the position
- Value – used as the information after determining the position

Attention Visualizations



Model Architecture of Transformer

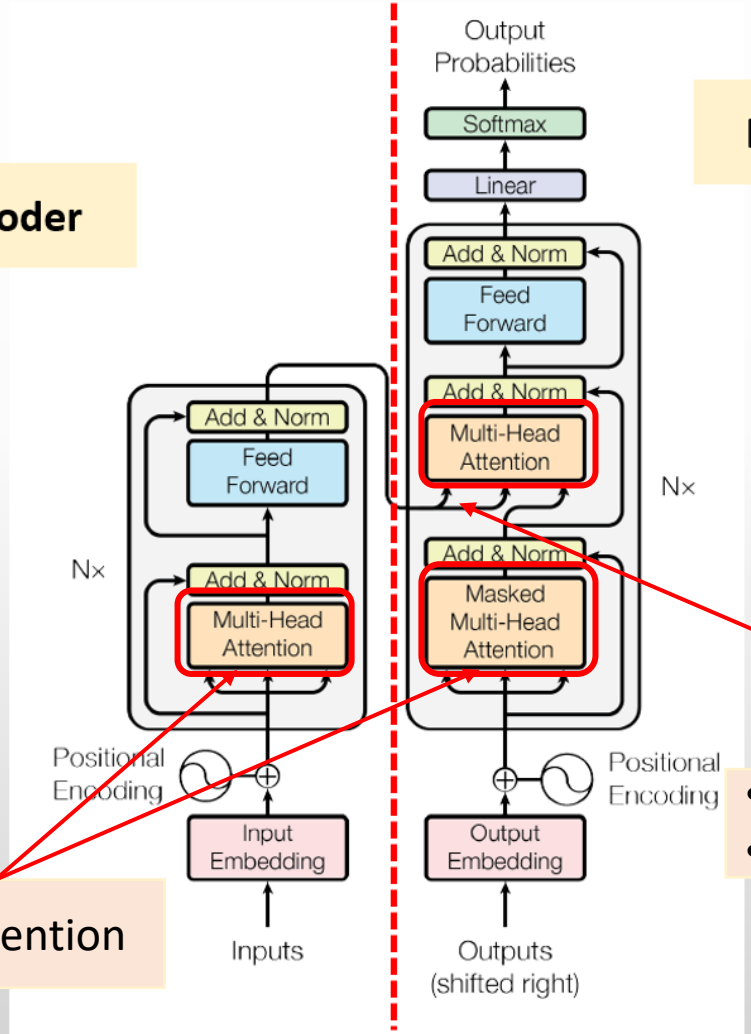


Figure 1: The Transformer - model architecture.

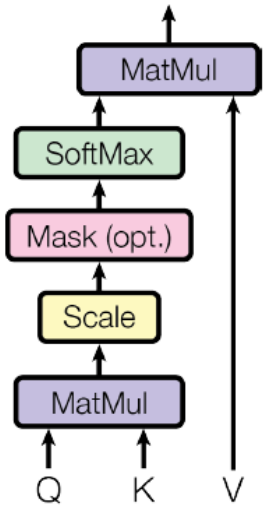
Encoder

Decoder

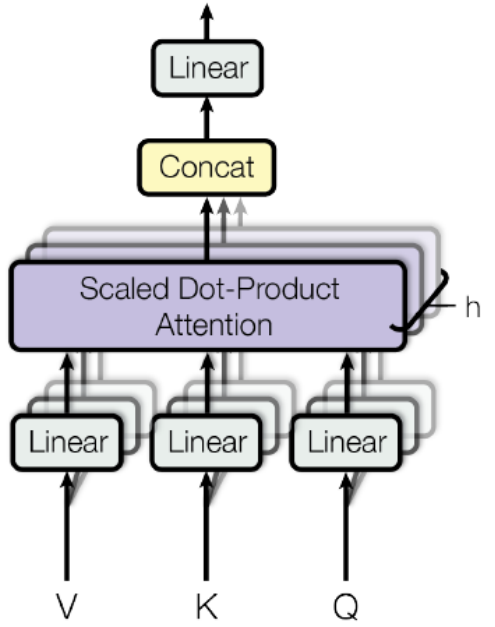
$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Scaled Dot-Product Attention



Multi-Head Attention



- Encoder gives Value and Key
- The decoder gives Query

Self attention

- Query – used with Key to determine the position
- Value – used as the information after determining the position

Model Architecture of Transformer

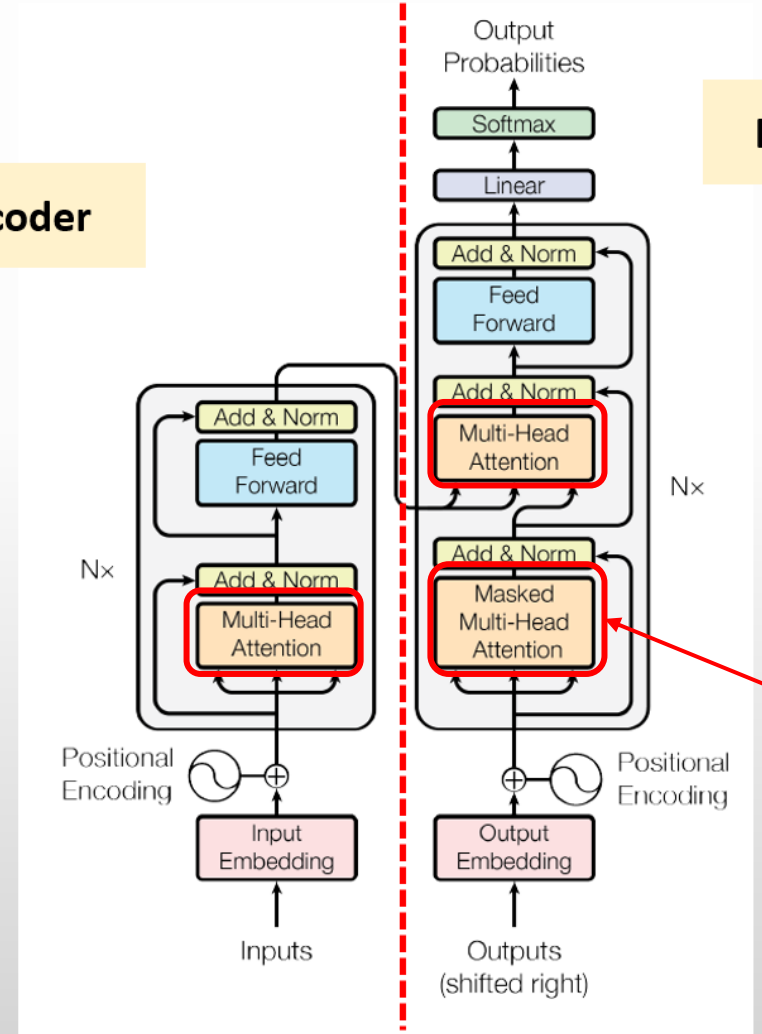
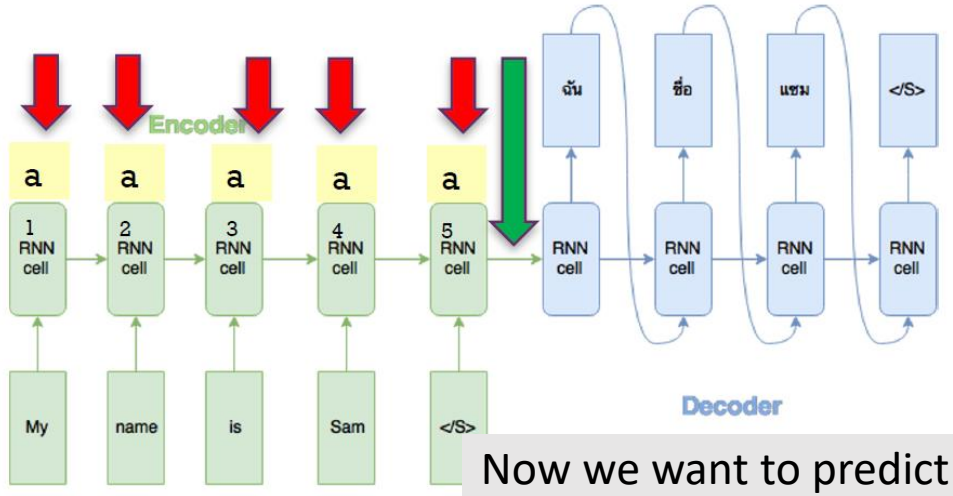


Figure 1: The Transformer - model architecture.

Encoder

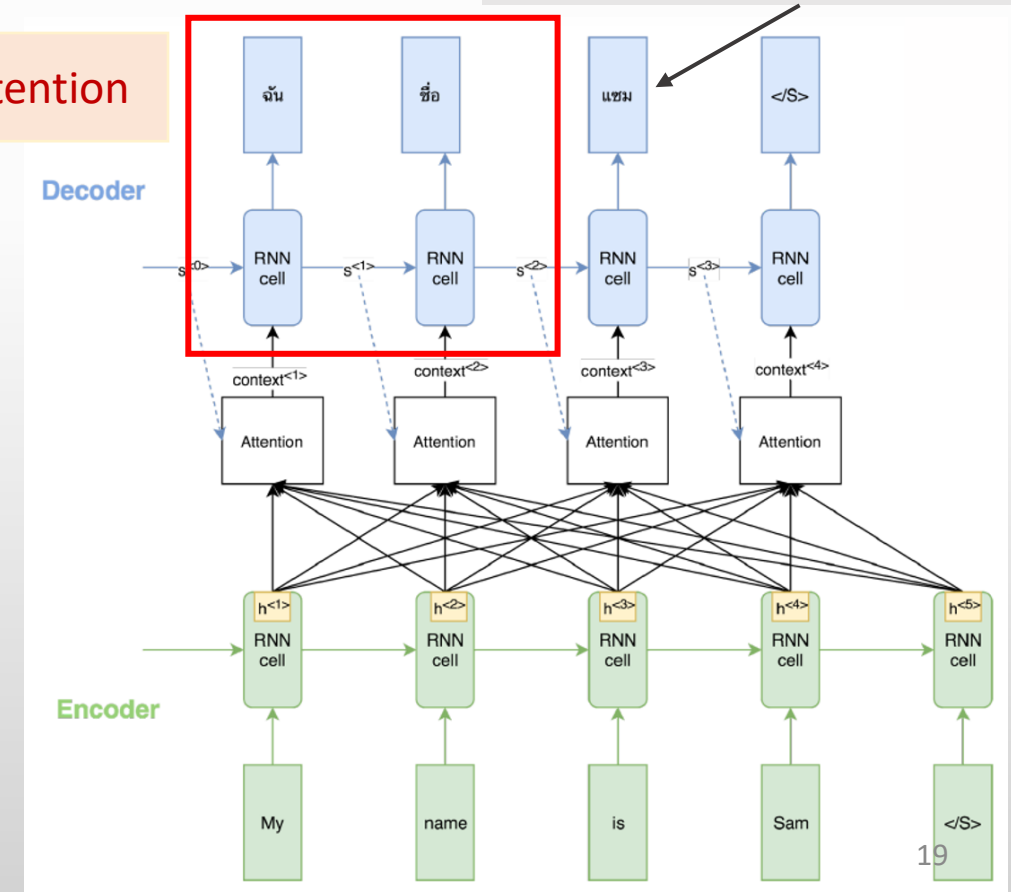
Decoder

A mask prevents looking ahead (cheating)



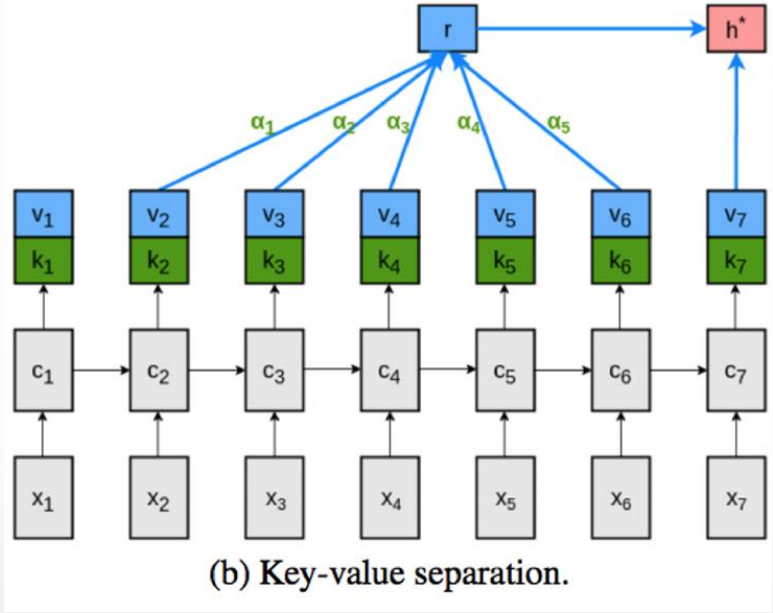
Now we want to predict "แซม"

Self attention

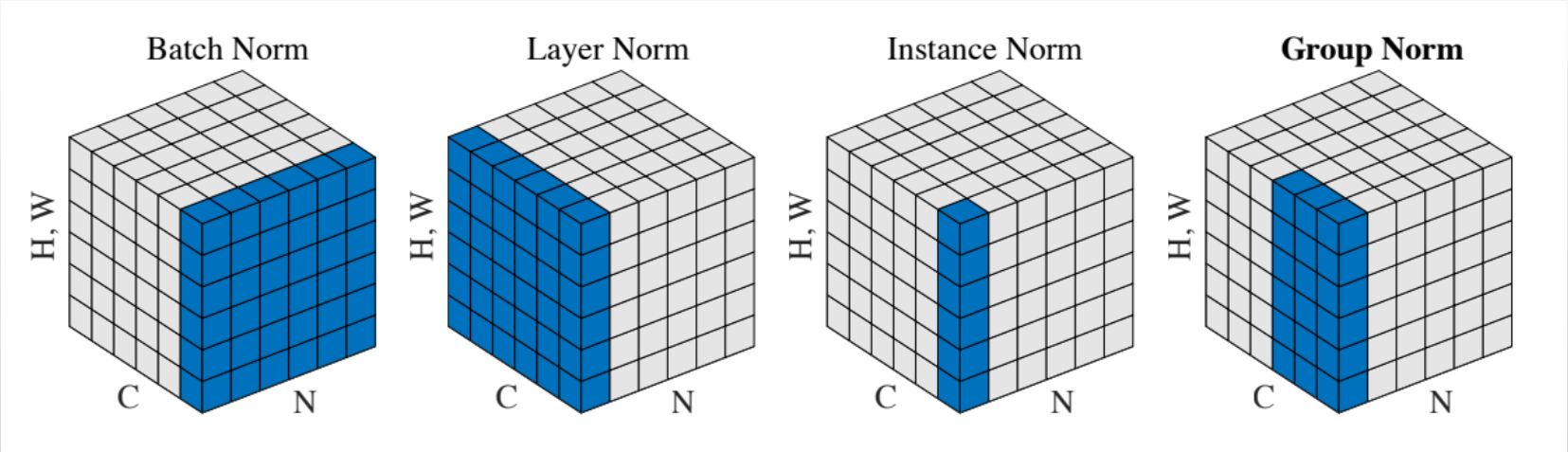


Model Architecture of Transformer

- **Layer norm**
- Normalize the mean and SD
- BN and GN are usually best, GN is better when batch size is small (Vision task)
- N – example in mini batch
- C – Channel output
- H,W – spatial coordinates (x,y)



(b) Key-value separation.



https://arxiv.org/pdf/1803.08494.pdf

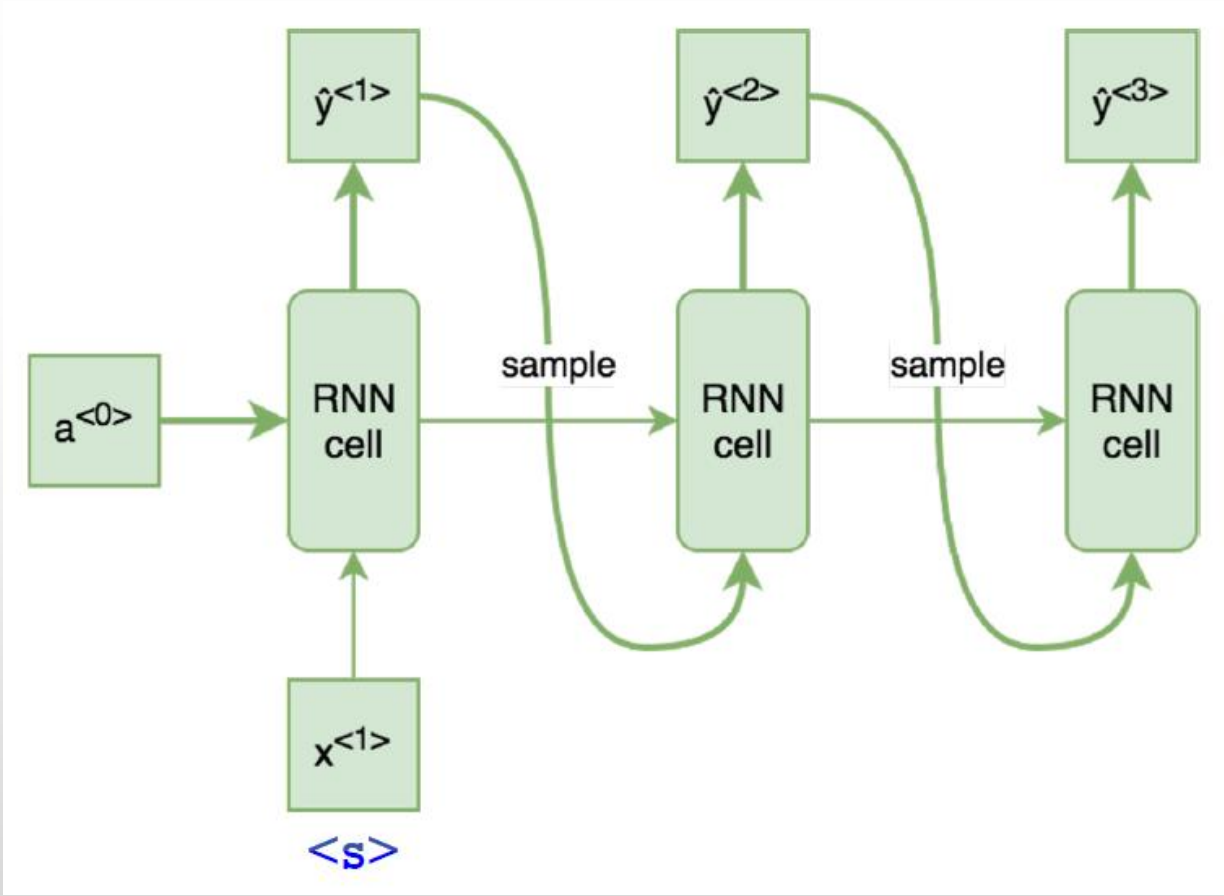
MT results with Transformer

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

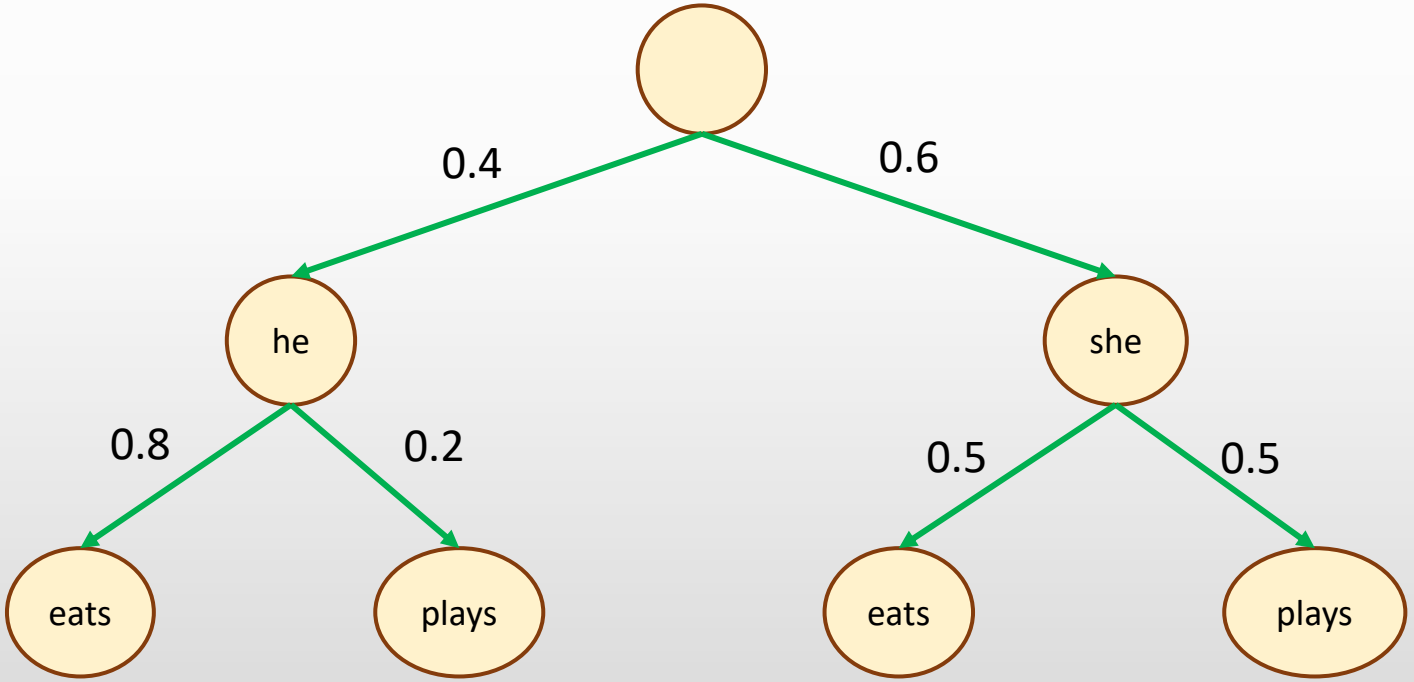
Text generation model (inference; testing)

- To generate a novel sequence, the inference model (testing phase) randomly samples an output from a softmax distribution.



Beam search

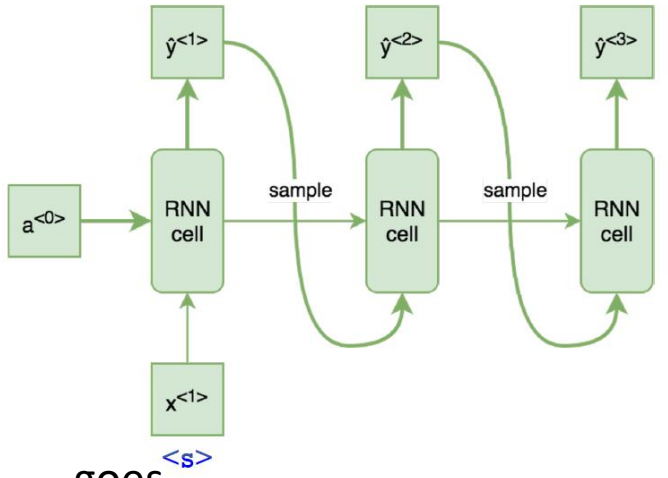
- Autoregressive requires making a decision even though it might not be optimal for the whole sequence
- However, expanding the tree will be exponential



- P(he eats) = 0.32
- P(she eats) = 0.30

Beam search

- Keep K active states at each step ; K = 2



softmax

he she

1 1 0 ... 0 0

$\hat{y}^{<1>}$

he → softmax

she → softmax

is goes

0 0 1 ... 1 0

is does

0 0 1 ... 0 1

$\hat{y}^{<2>}$

---

he is → softmax

a very

0 0 0

he goes → softmax

0 0 0

$\hat{y}^{<3>}$

he is a → softmax

0 0 0

he is very → softmax

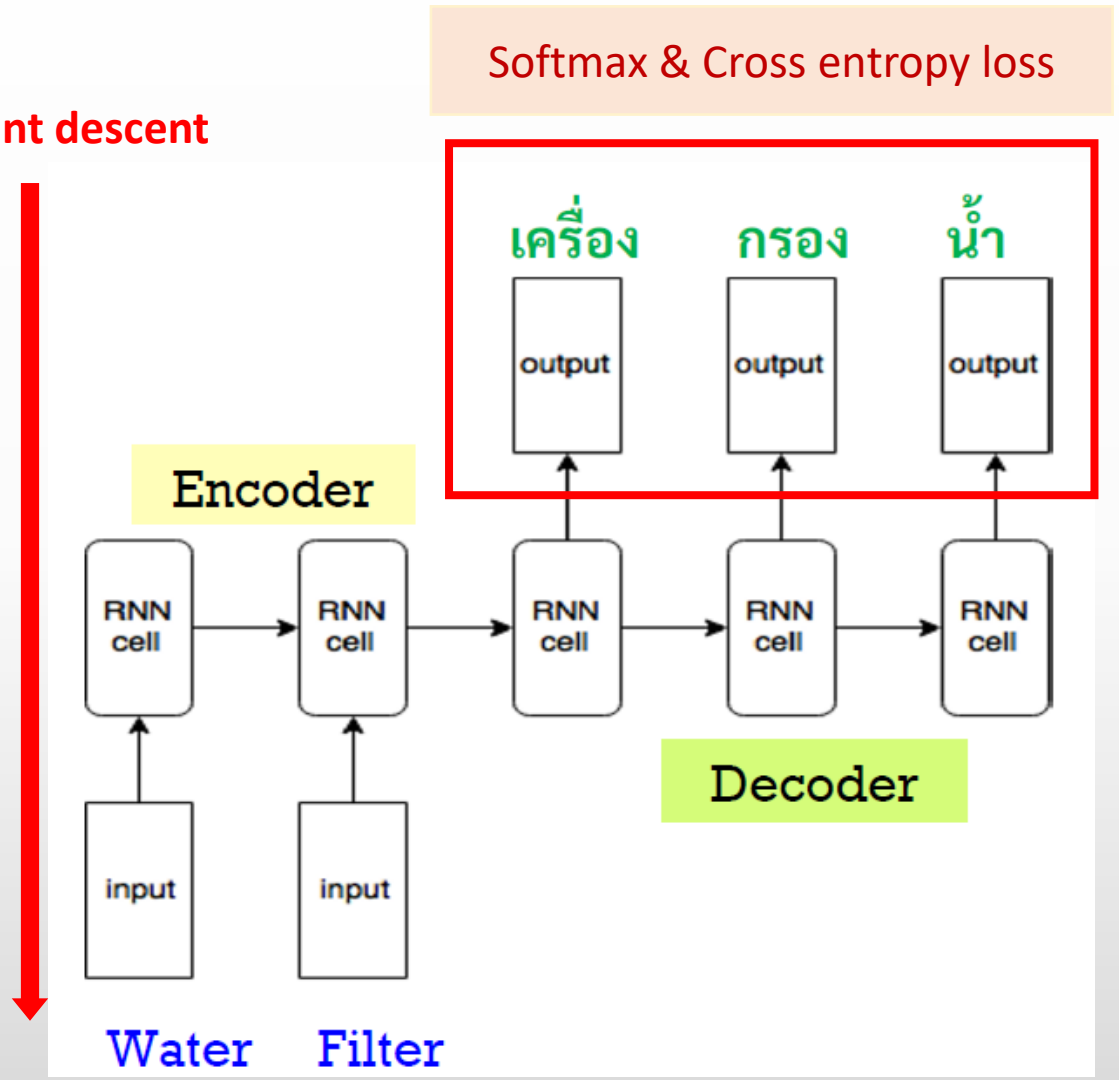
0 0 0

$\hat{y}^{<3>}$

...

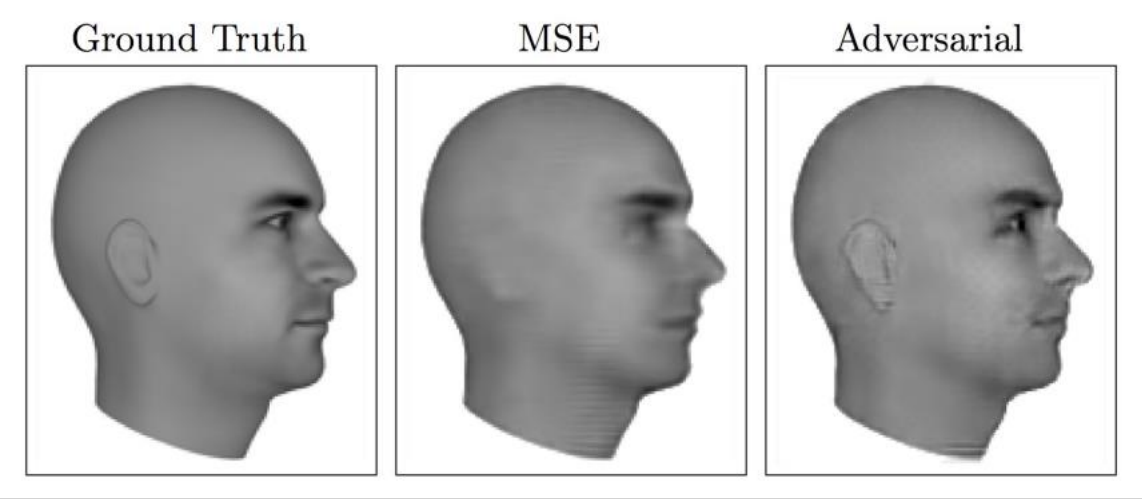
RNN architectures in NPL

- **Many-to-many (encoder-decoder)**
- Sequence Input, Sequence output
- These two sequences can be of different length
- E.g. Machine Translation
  - Input: English Sentence
  - Output: Thai Sentence
- Machine Translation is also a text generation task



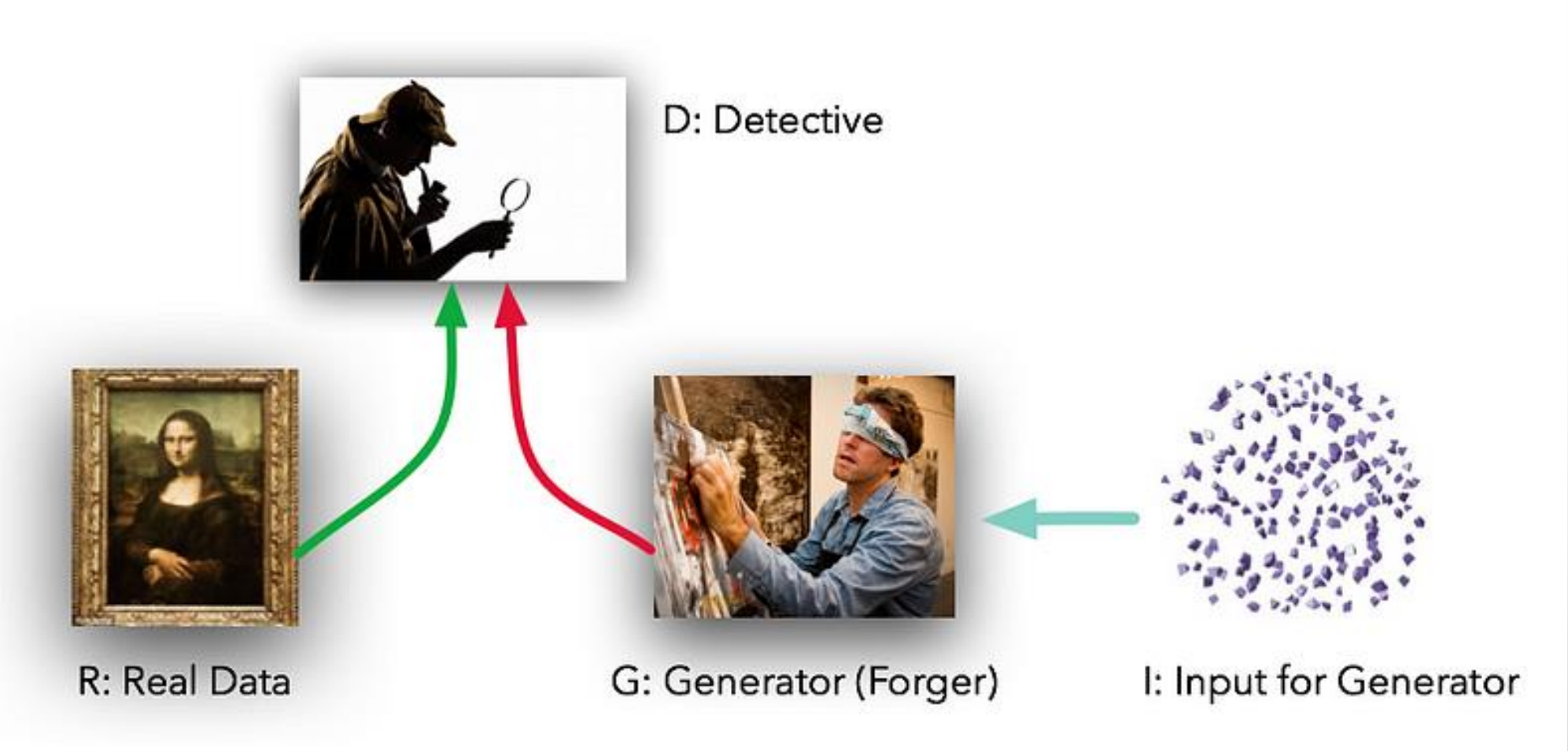
Generative Adversarial Networks (GANs)

- Learning distributions
- Supervised learning tasks usually have one correct answer
- Sometimes there are more than one possibility
  - What is the next frame of a video?
  - What is the missing pixels in an image?
  - What word is missing from the blank?
    - I eat ___



https://arxiv.org/pdf/1701.00160.pdf

Generative Adversarial Networks (GANs)



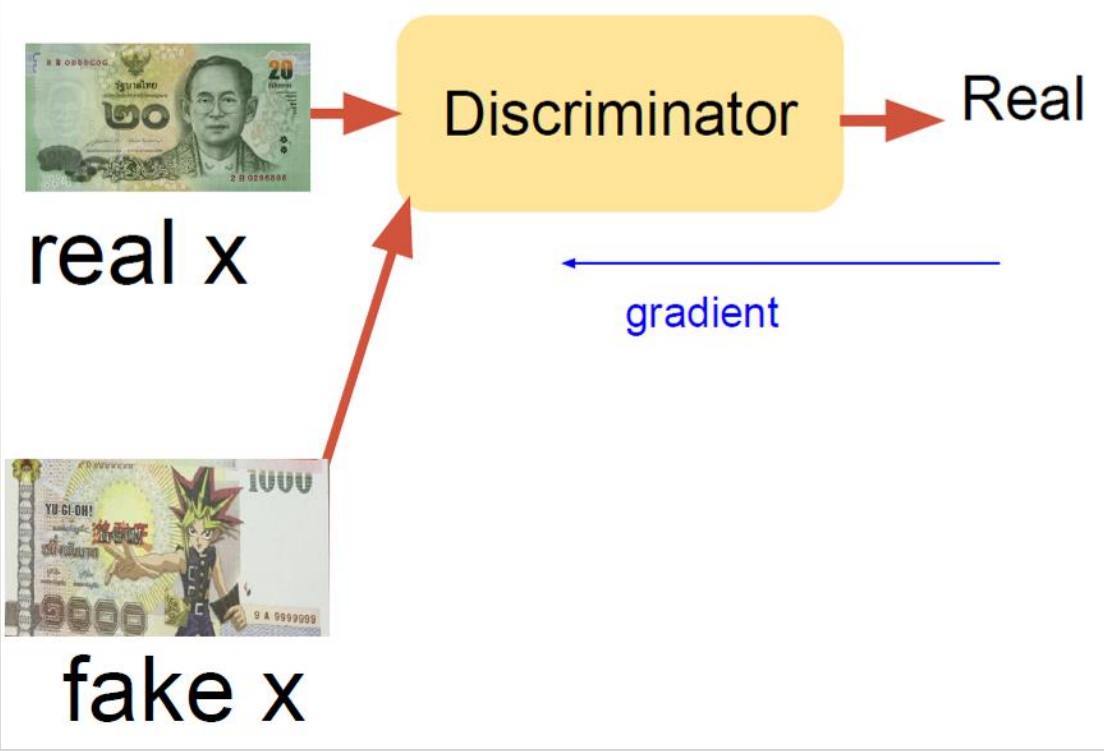
- https://medium.com/@devnag/generative-adversarial-networks-gans-in-50-lines-of-code-pytorch-e81b79659e3f

Generative Adversarial Networks (GANs)

- Consider a money counterfeiter
  - He wants to make fake money that looks real
  - There's a police that tries to differentiate fake and real money.
- The counterfeiter is the adversary and is generating fake inputs. – Generator network
- The police is try to discriminate between fake and real inputs. – Discriminator network

Generative Adversarial Networks (GANs)

- Train Discriminator
  - The discriminator learns to differentiate real and fake data
  - Learns by backpropagation



Generative Adversarial Networks (GANs)

- Train Generator
  - The generator learns to be better by the gradient given by the discriminator

Generative Adversarial Networks (GANs)

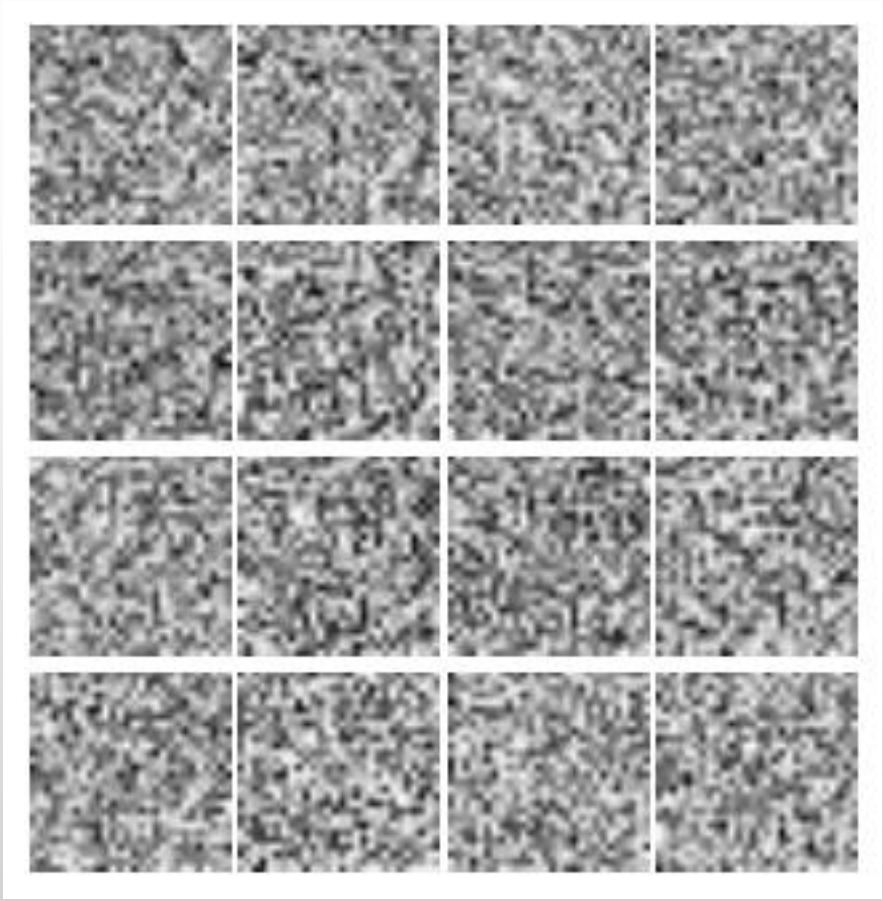
- Generator (Money Faker): $\min_G \mathbb{E}_{\mathbf{z}}[\log(1 - D(G(\mathbf{z}))]$ $\mathcal{L}_G = -\frac{1}{m}\sum_{i=1}^{m} log(D(G(z^{(i)})))$
  - Maximize Y
- Discriminator (Police):
  - For real images => Maximize Y
  - For generated images from the faker => Minimize Y

$\max_D \mathbb{E}_{\mathbf{z}}[\log(1 - D(G(\mathbf{z}))] + \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[\log(D(\mathbf{x}))]$

$\mathcal{L}_D = -\frac{1}{m}\sum_{i=1}^{m}(log(D(x^{(i)})) + log(1 - D(G(z^{(i)}))))$

Generative Adversarial Networks (GANs)

- Generator output starts from random noise and gets better as we train.



https://agustinus.kristia.de/techblog/2016/09/17/gan-tensorflow/

GANs Loss Formulations

Discriminator

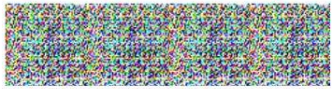
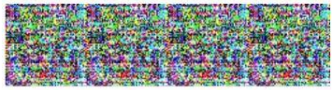
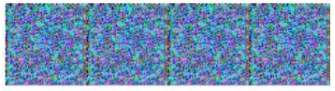
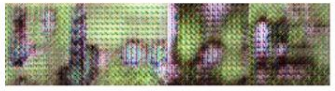
$$\max_D \mathbb{E}_{\mathbf{z}}[\log(1 - D(G(\mathbf{z}))] + \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[\log(D(\mathbf{x}))]$$

Generator

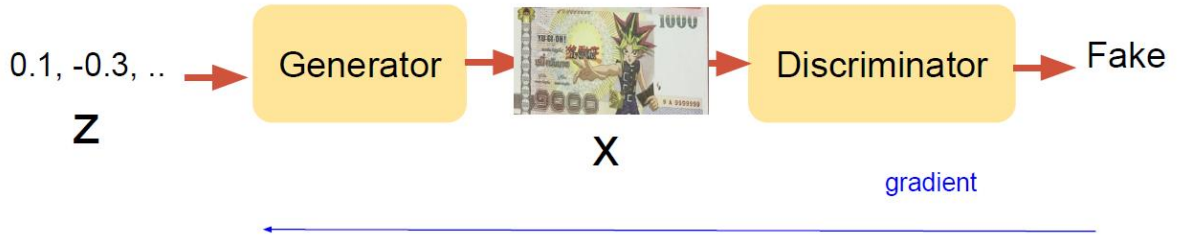
$$\min_G \mathbb{E}_{\mathbf{z}}[\log(1 - D(G(\mathbf{z}))]$$

GAN	Discriminator Loss	Generator Loss
MM GAN	$\mathcal{L}_{\text{D}}^{\text{GAN}} = -\mathbb{E}_{x\sim p_d}[\log(D(x))] - \mathbb{E}_{\hat{x}\sim p_g}[\log(1 - D(\hat{x}))]$	$\mathcal{L}_{\text{G}}^{\text{GAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\log(1 - D(\hat{x}))]$
NS GAN	$\mathcal{L}_{\text{D}}^{\text{NSGAN}} = -\mathbb{E}_{x\sim p_d}[\log(D(x))] - \mathbb{E}_{\hat{x}\sim p_g}[\log(1 - D(\hat{x}))]$	$\mathcal{L}_{\text{G}}^{\text{NSGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[\log(D(\hat{x}))]$
WGAN	$\mathcal{L}_{\text{D}}^{\text{WGAN}} = -\mathbb{E}_{x\sim p_d}[D(x)] + \mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$	$\mathcal{L}_{\text{G}}^{\text{WGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$
WGAN GP	$\mathcal{L}_{\text{D}}^{\text{WGANGP}} = \mathcal{L}_{\text{D}}^{\text{WGAN}} + \lambda\mathbb{E}_{\hat{x}\sim p_g}[(\|\|\nabla D(\alpha x + (1 - \alpha\hat{x})\|\|_2 - 1)^2]$	$\mathcal{L}_{\text{G}}^{\text{WGANGP}} = -\mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})]$
LS GAN	$\mathcal{L}_{\text{D}}^{\text{LSGAN}} = -\mathbb{E}_{x\sim p_d}[(D(x) - 1)^2] + \mathbb{E}_{\hat{x}\sim p_g}[D(\hat{x})^2]$	$\mathcal{L}_{\text{G}}^{\text{LSGAN}} = -\mathbb{E}_{\hat{x}\sim p_g}[(D(\hat{x} - 1))^2]$
DRAGAN	$\mathcal{L}_{\text{D}}^{\text{DRAGAN}} = \mathcal{L}_{\text{D}}^{\text{GAN}} + \lambda\mathbb{E}_{\hat{x}\sim p_d + \mathcal{N}(0,c)}[(\|\|\nabla D(\hat{x})\|\|_2 - 1)^2]$	$\mathcal{L}_{\text{G}}^{\text{DRAGAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\log(1 - D(\hat{x}))]$
BEGAN	$\mathcal{L}_{\text{D}}^{\text{BEGAN}} = \mathbb{E}_{x\sim p_d}[\|\|x - \text{AE}(x)\|\|_1] - k_t\mathbb{E}_{\hat{x}\sim p_g}[\|\|\hat{x} - \text{AE}(\hat{x})\|\|_1]$	$\mathcal{L}_{\text{G}}^{\text{BEGAN}} = \mathbb{E}_{\hat{x}\sim p_g}[\|\|\hat{x} - \text{AE}(\hat{x})\|\|_1]$

GANs Loss Formulations

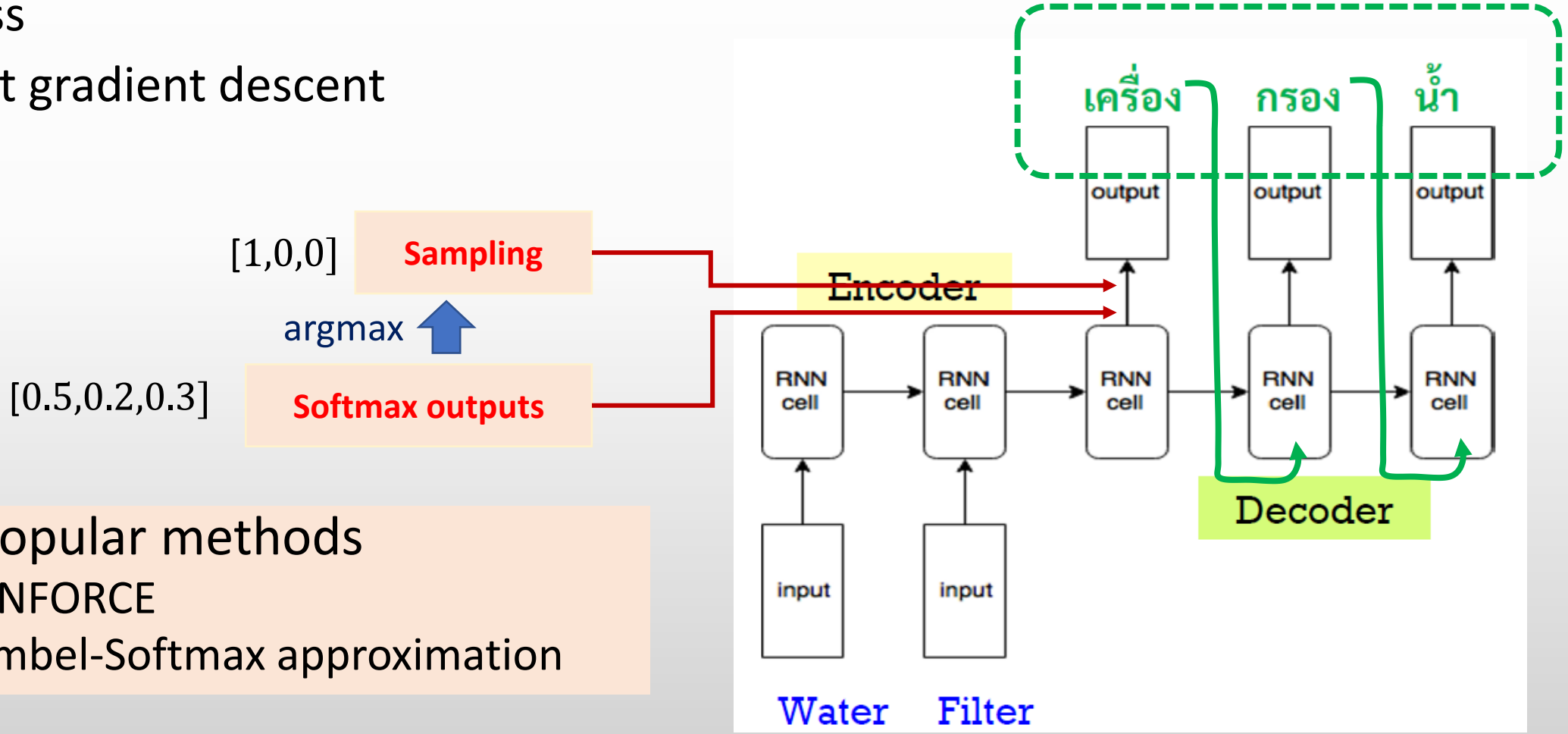
DCGAN	LSGAN	WGAN (clipping)	WGAN-GP (ours)
Baseline (G: DCGAN, D: DCGAN)			
G: No BN and a constant number of filters, D: DCGAN			
G: 4-layer 512-dim ReLU MLP, D: DCGAN			
No normalization in either G or D			
Gated multiplicative nonlinearities everywhere in G and D			
tanh nonlinearities everywhere in G and D			
101-layer ResNet G and D			

GANs for text generatior



- Autoregressive decoding includes a sampling process
- Cannot gradient descent

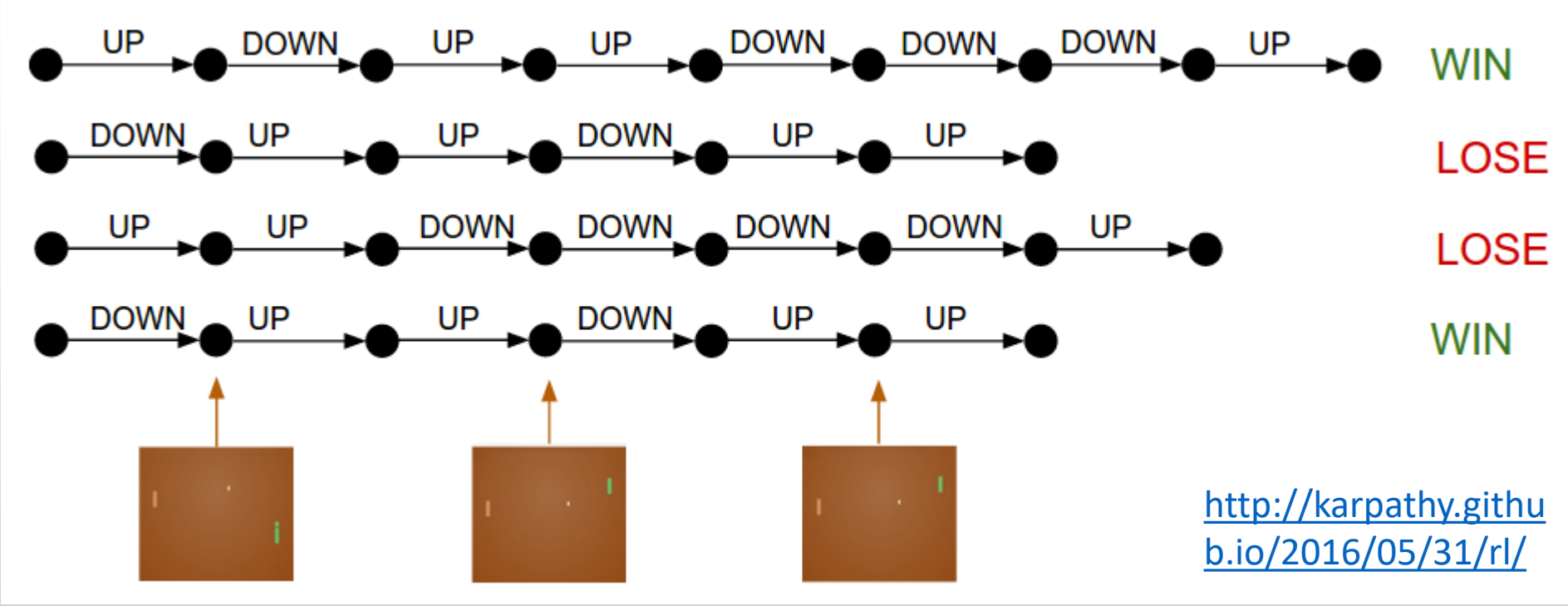
Discriminator sees the sampled sentence.
Says it's good or bad



- Two popular methods
  - REINFORCE
  - Gumbel-Softmax approximation

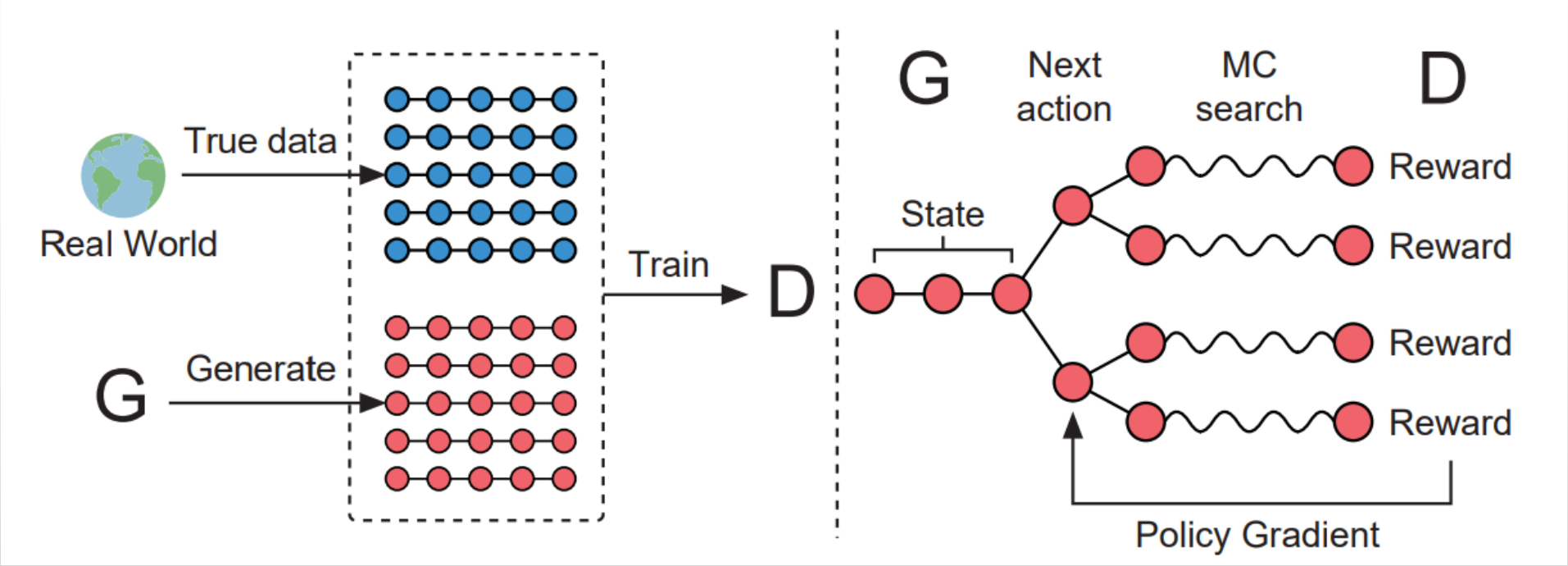
RL and policy gradients

- Credit assignment problem in reinforcement learning
- Which move makes you win?
- For RL with policy gradients, we increase the probability of every move that results in a win (REINFORCE algorithm)



GAN with text generation(SeqGAN)

- Use policy gradient to update the generator (the agent in RL setting)
- The discriminator (critic) gives the reward



https://arxiv.org/pdf/1609.05473.pdf

Gumbel Softmax

- From softmax prob [0.3 0.4 0.1 0.2] -> word index z
- Pick randomly based on prob (sampling)

$$z = \texttt{one\_hot}\left(\arg\max_i \left[g_i + \log \pi_i\right]\right)$$

random value generated from Gumbel dist.

index for each word

prob values from softmax

https://arxiv.org/pdf/1611.01144.pdf

https://arxiv.org/pdf/1611.01144.pdf

Gumbel Softmax

- From softmax prob [0.3 0.4 0.1 0.2] -> word index z
- Pick randomly based on prob (sampling)

random value generated from Gumbel dist.

$$z = \text{one\_hot}\left(\arg\max_i \left[g_i + \log \pi_i\right]\right)$$

index for each word

prob values from softmax

estimate using Gumbel trick

$$y_i = \frac{\exp((\log(\pi_i) + g_i)/\tau)}{\sum_{j=1}^{k} \exp((\log(\pi_j) + g_j)/\tau)} \quad \text{for } i = 1, ..., k.$$

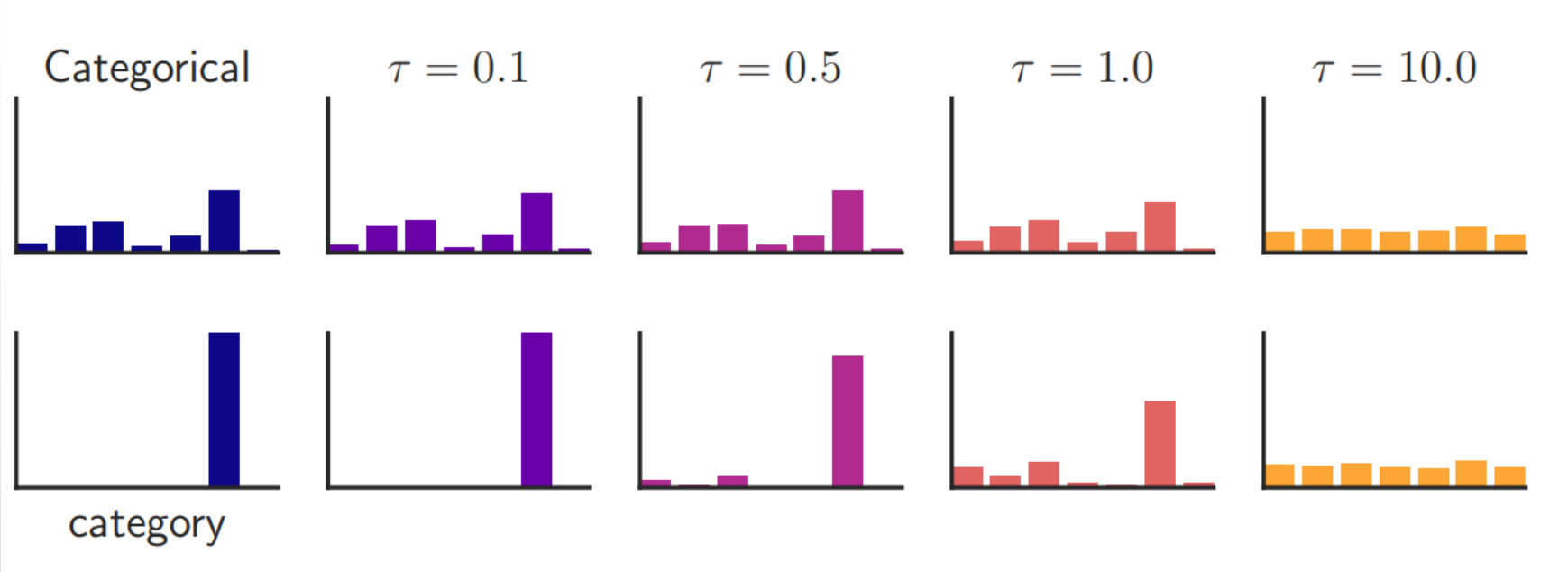
Temperature parameter

https://arxiv.org/pdf/1611.01144.pdf

Gumbel Softmax

$$y_i = \frac{\exp((\log(\pi_i) + g_i)/\tau)}{\sum_{j=1}^{k} \exp((\log(\pi_j) + g_j)/\tau)} \quad \text{for } i = 1, ..., k.$$

- Temperature can be scaled
- y can be back propagated through



HW4

https://drive.google.com/drive/folders/13nW1B1DGkwRrUjZfsEcrKDiG8MCaE6mU?usp=share_link