UNIVERSIDAD EAFIT

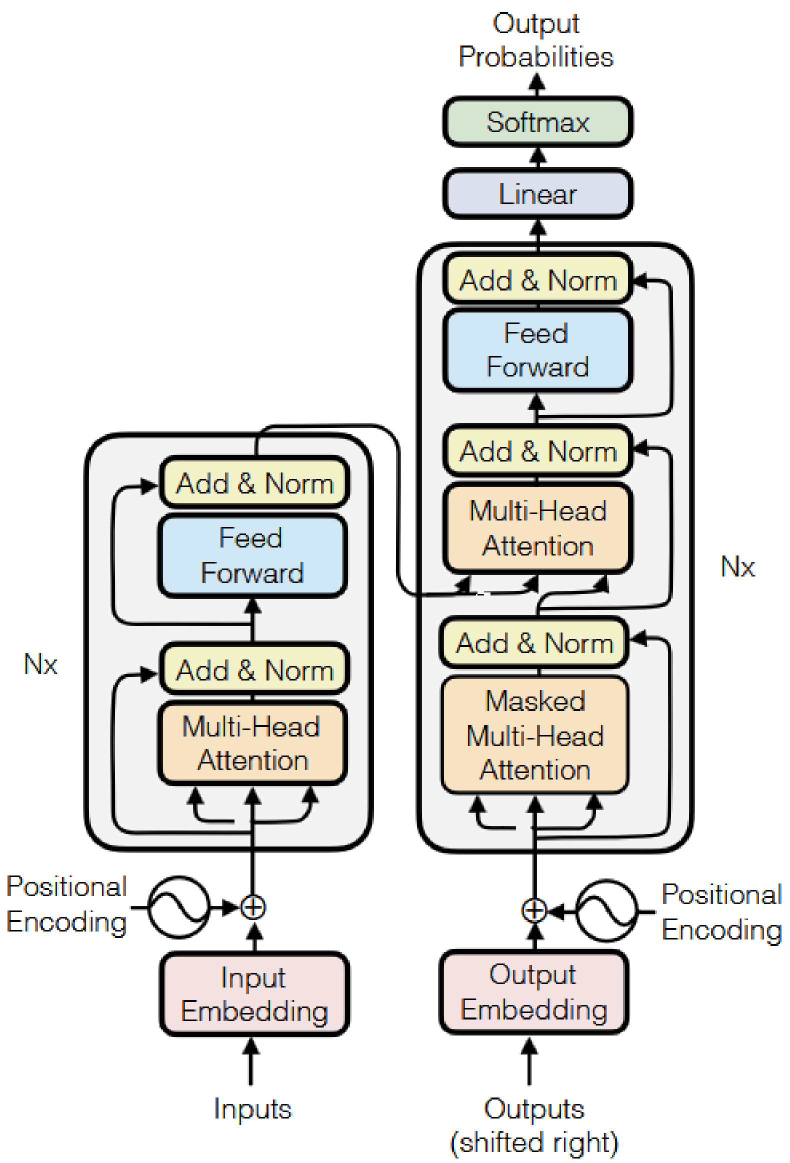
Escuela de Ciencias Aplicadas e Ingeniería

Artificial Intelligence

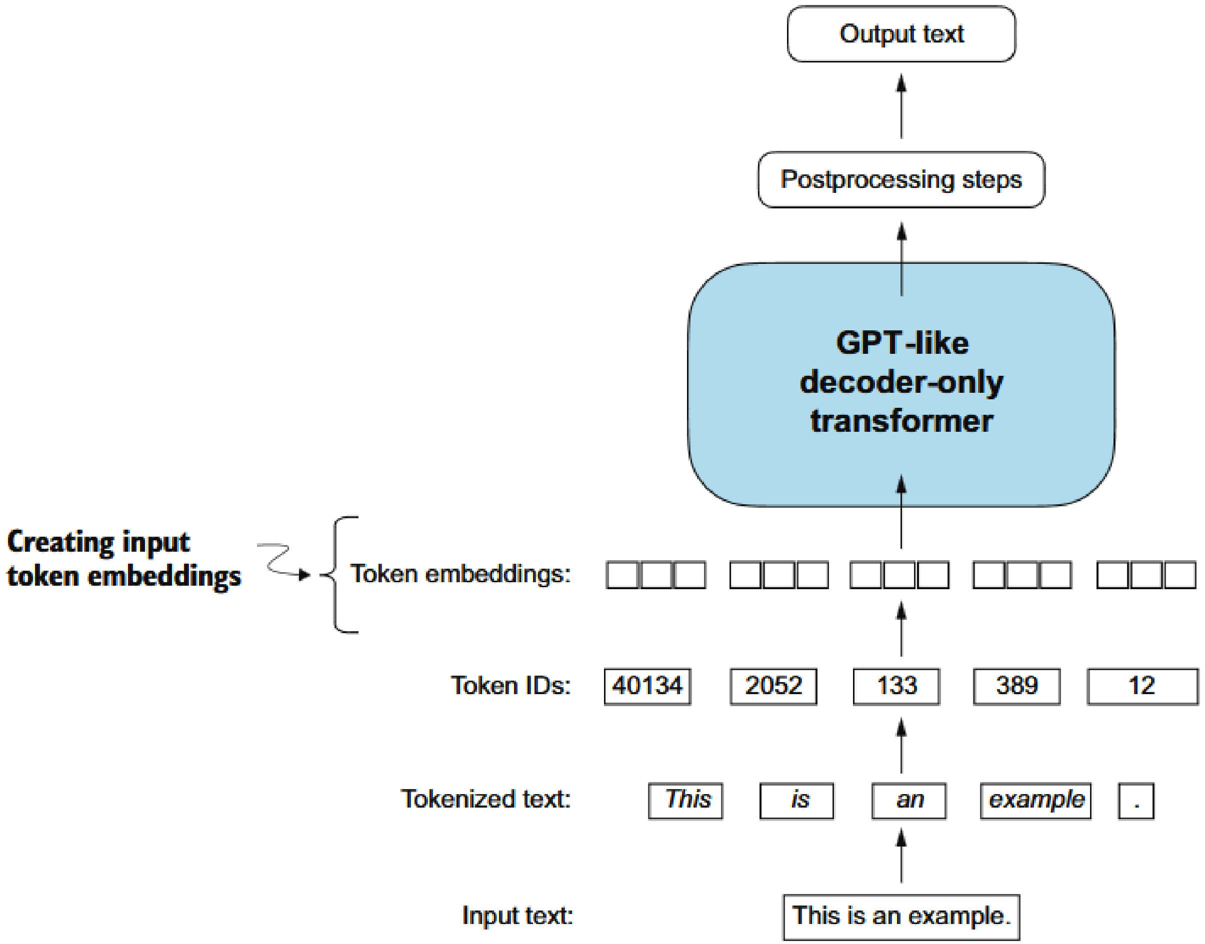
Lecture06a – Attention is all you need

Contenido

1. Token embeddings
2. Positional encodings
3. Sequence modeling
4. Recurrent neural networks
5. Attention


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

Creating token embeddings



Creating token embeddings

Embedding layers: An efficient way of performing matrix multiplication when working with one-hot encoded vectors.

```
import torch

torch.manual_seed(123);
```

```
idx = torch.tensor([2, 3, 1]) # 3 training examples

num_idx = max(idx)+1
out_dim = 5
```

Input dimension of a one-hot encoded vector is the number of indices (the highest index + 1)

Suppose we want embeddings of size 5

Creating token embeddings

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

```
import torch

torch.manual_seed(123);
```

```
idx = torch.tensor([2, 3, 1]) # 3 training examples

num_idx = max(idx)+1
out_dim = 5

embedding = torch.nn.Embedding(num_idx, out_dim)
embedding(idx)
```

```
tensor([[ 0.6957, -1.8061, -1.1589,  0.3255, -0.6315],
        [-2.8400, -0.7849, -1.4096, -0.4076,  0.7953],
        [ 1.3010,  1.2753, -0.2010, -0.1606, -0.4015]],
       grad_fn=<EmbeddingBackward0>)
```

Each training example has 5 feature values

Creating token embeddings

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

One-hot encoded ("sparse") representation of "S U N N Y"

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
S	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
U	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
N	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
N	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Y	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

↓

Embedded ("dense") representation of "S U N N Y"

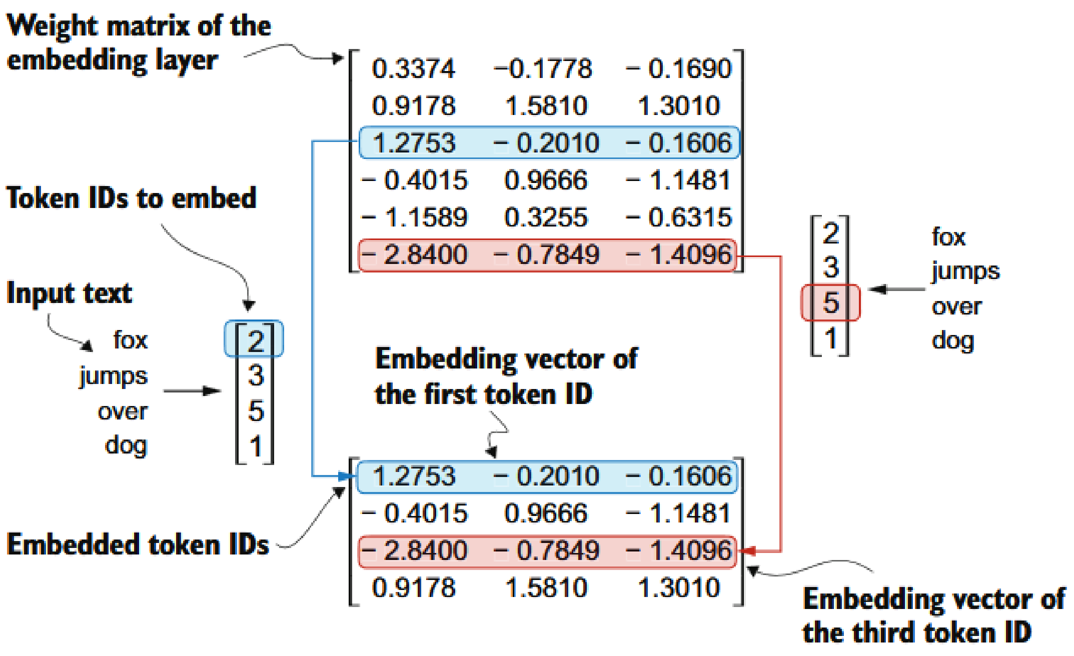
```
[[0.9816, 0.7363, 0.5899],
 [0.2605, 0.3766, 0.3502],
 [0.7382, 0.9807, 0.4762],
 [0.6231, 0.8825, 0.8836]]
```

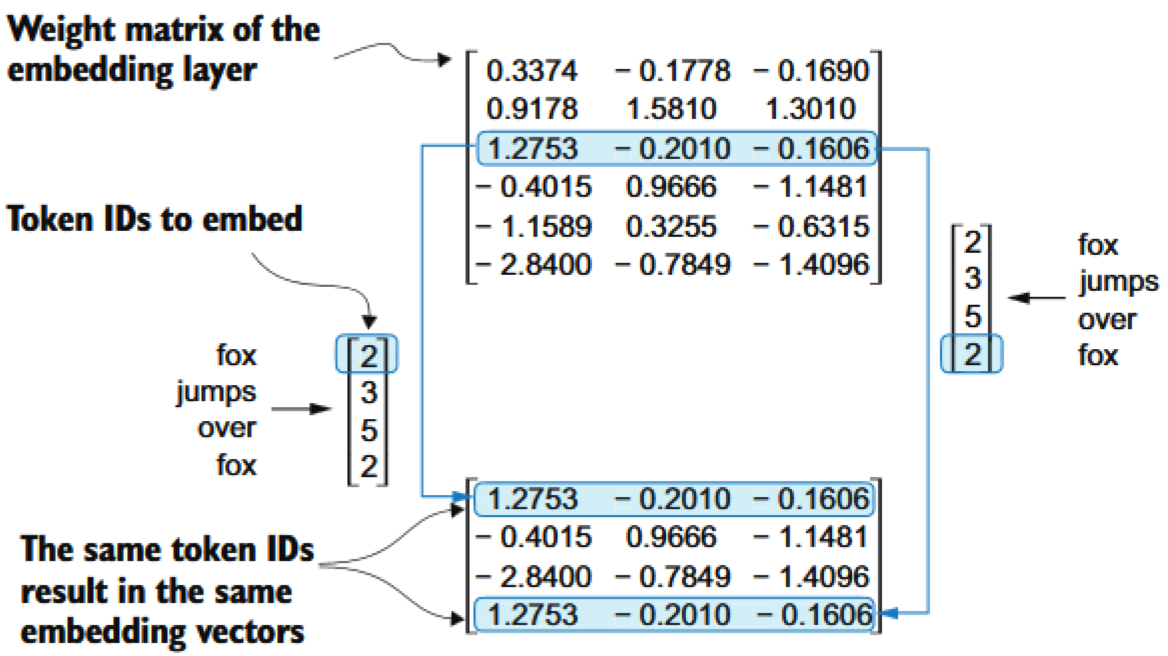
Embedding layer

```
  [[0.6912, 0.8765, 0.4939],
   [0.6342, 0.7481, 0.7717],
   [0.8395, 0.2128, 0.3696],
   [0.4900, 0.1509, 0.0689],
   [0.2587, 0.9171, 0.8670],
   [0.7213, 0.9922, 0.5701],
   [0.7598, 0.5231, 0.3666],
   [0.5150, 0.5216, 0.9682],
   [0.2248, 0.0261, 0.4427],
   [0.1818, 0.6863, 0.8713],
   [0.4192, 0.1566, 0.9004],
   [0.8102, 0.5741, 0.4241],
   [0.1116, 0.0466, 0.2786],
S  [0.9816, 0.7363, 0.5899],
   [0.9224, 0.3672, 0.6972],
   [0.1207, 0.3372, 0.2128],
   [0.0660, 0.1524, 0.8440],
   [0.2162, 0.5640, 0.0988],
U  [0.2605, 0.3766, 0.3502],
   [0.2334, 0.4757, 0.7581],
N  [0.7382, 0.9807, 0.4762],
   [0.2369, 0.8102, 0.8798],
   [0.6932, 0.2671, 0.8018],
   [0.9593, 0.5302, 0.4290],
Y  [0.6231, 0.8825, 0.8836],
   [0.4623, 0.8503, 0.7279]]
```

Creating token embeddings

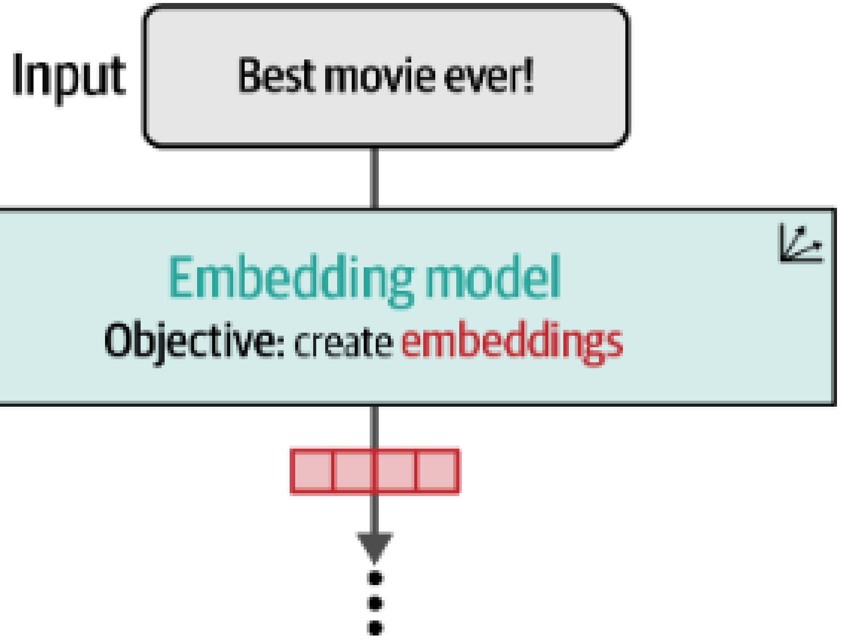
UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería



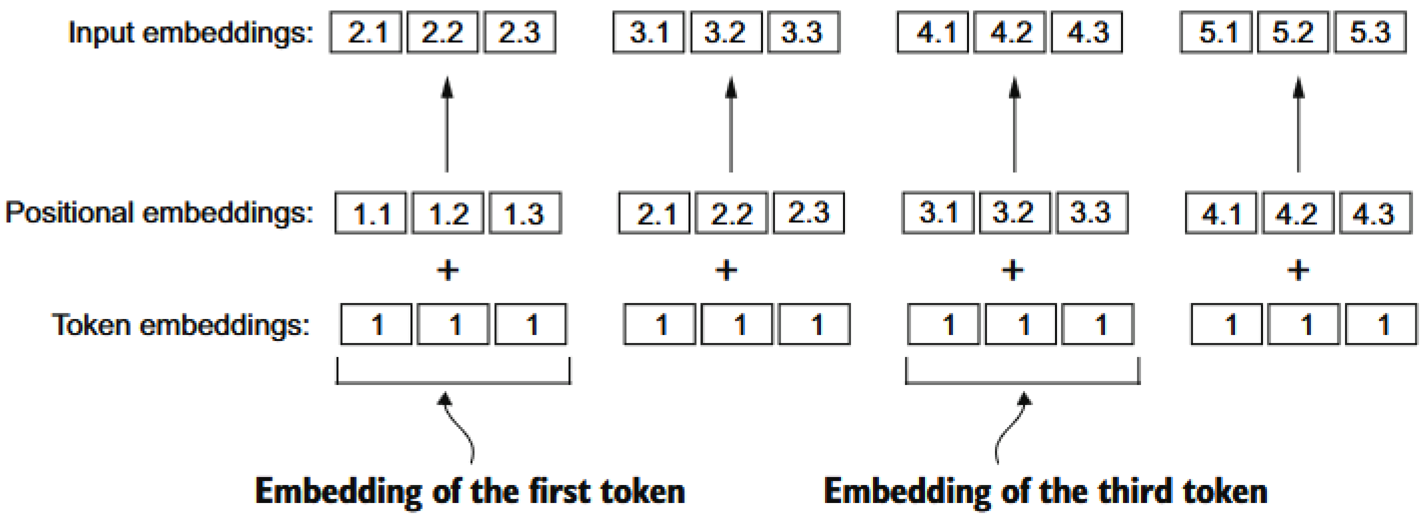


Sentence embeddings

Task: Aggregate token embeddings in a sentence to form a **unique sentence representation.**



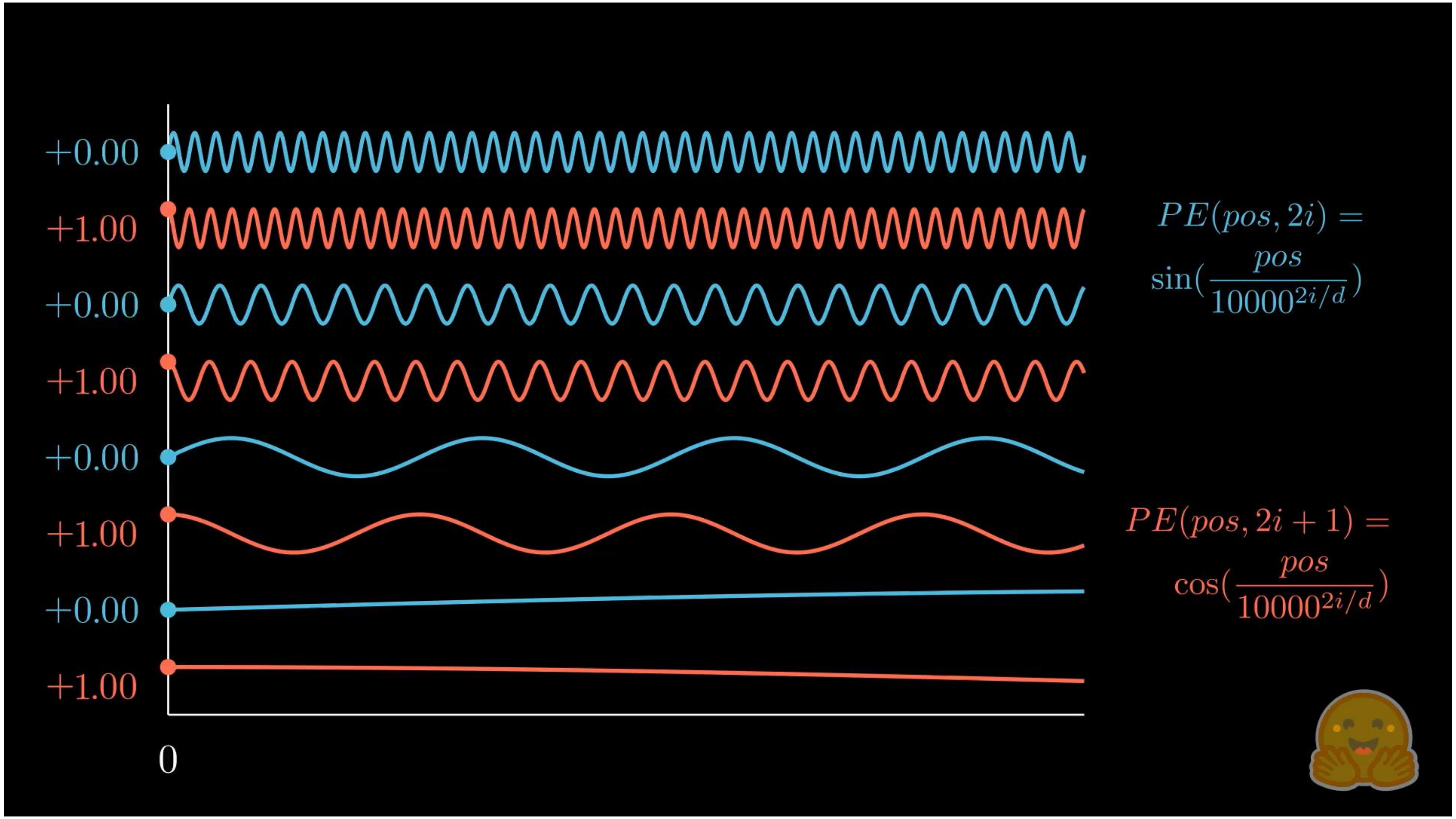
Encoding word positions



OpenAI: Positional embeddings are learnable parameters.
Llama, DeepSeek: Rotational Positional Embeddings – RoPE
Bert: Sinusoidal Positional Embeddings

Sinusoidal positional embeddings

UNIVERSIDAD EAFIT | Escuela de Ciencias Aplicadas e Ingeniería



$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

$$PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

Rotary Positional Embeddings (RoPE)

Fundamental idea: we want the dot product of embeddings to result in a function of relative position:

$$f_q(\mathbf{x}_m, m) \cdot f_k(\mathbf{x}_n, n) = g(\mathbf{x}_m, \mathbf{x}_n, m - n)$$

ROFORMER: ENHANCED TRANSFORMER WITH ROTARY POSITION EMBEDDING

Jianlin Su Zhuiyi Technology Co., Ltd. Shenzhen bojonesu@wezhuiyi.com
Yu Lu Zhuiyi Technology Co., Ltd. Shenzhen julianlu@wezhuiyi.com
Shengfeng Pan Zhuiyi Technology Co., Ltd. Shenzhen nickpan@wezhuiyi.com
Ahmed Murtadha Zhuiyi Technology Co., Ltd. Shenzhen mengjiayi@wezhuiyi.com
Bo Wen Zhuiyi Technology Co., Ltd. Shenzhen brucewen@wezhuiyi.com
Yunfeng Liu Zhuiyi Technology Co., Ltd. Shenzhen glenliu@wezhuiyi.com

In summary, RoPE uses trigonometry to come up with a function that satisfies this property

$$R^d_{\Theta,m}\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ \vdots \\ x_{d-1} \\ x_d \end{pmatrix} \otimes \begin{pmatrix} \cos m\theta_1 \\ \cos m\theta_1 \\ \cos m\theta_2 \\ \cos m\theta_2 \\ \vdots \\ \cos m\theta_{\frac{d}{2}} \\ \cos m\theta_{\frac{d}{2}} \end{pmatrix} + \begin{pmatrix} -x_2 \\ x_1 \\ -x_4 \\ x_3 \\ \vdots \\ -x_d \\ x_{d-1} \end{pmatrix} \otimes \begin{pmatrix} \sin m\theta_1 \\ \sin m\theta_1 \\ \sin m\theta_2 \\ \sin m\theta_2 \\ \vdots \\ \sin m\theta_{\frac{d}{2}} \\ \sin m\theta_{\frac{d}{2}} \end{pmatrix}$$

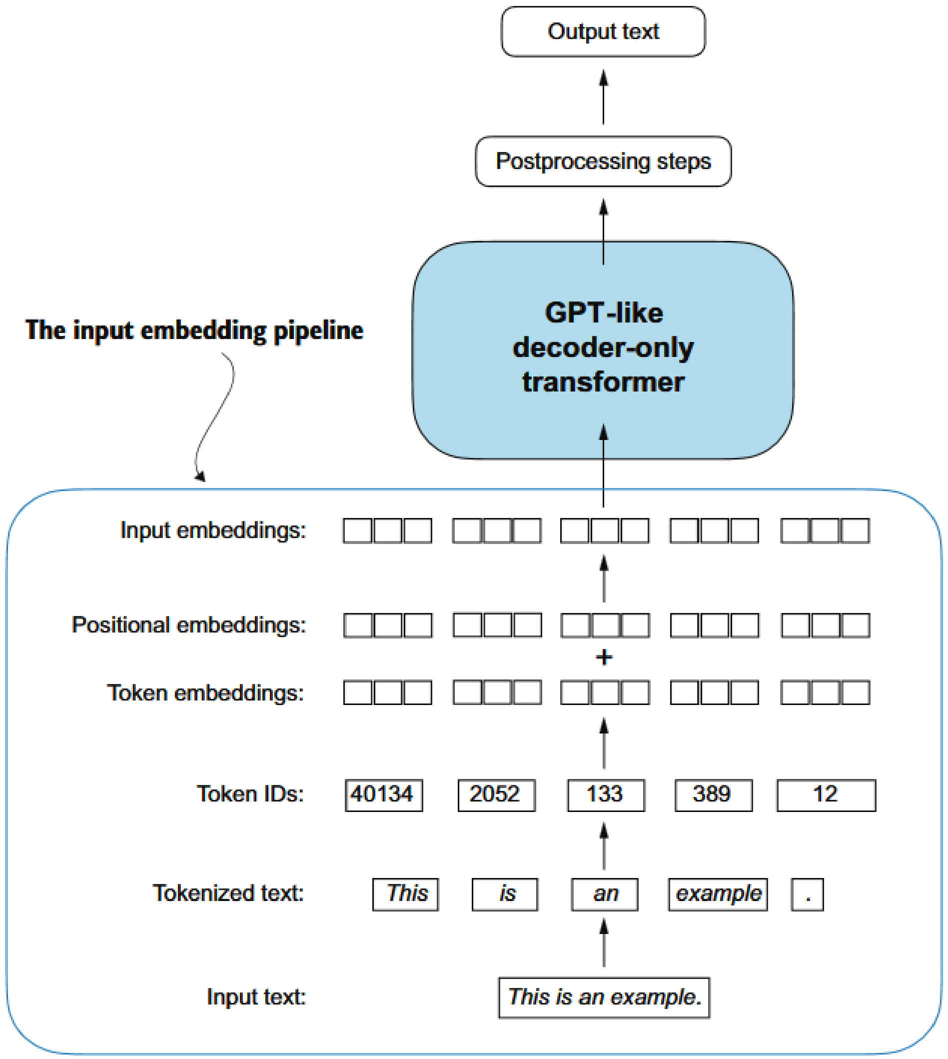
m is the token's position in the sequence.

$\Theta = (\theta_1, \theta_2, ..., \theta_{d/2})$ with:

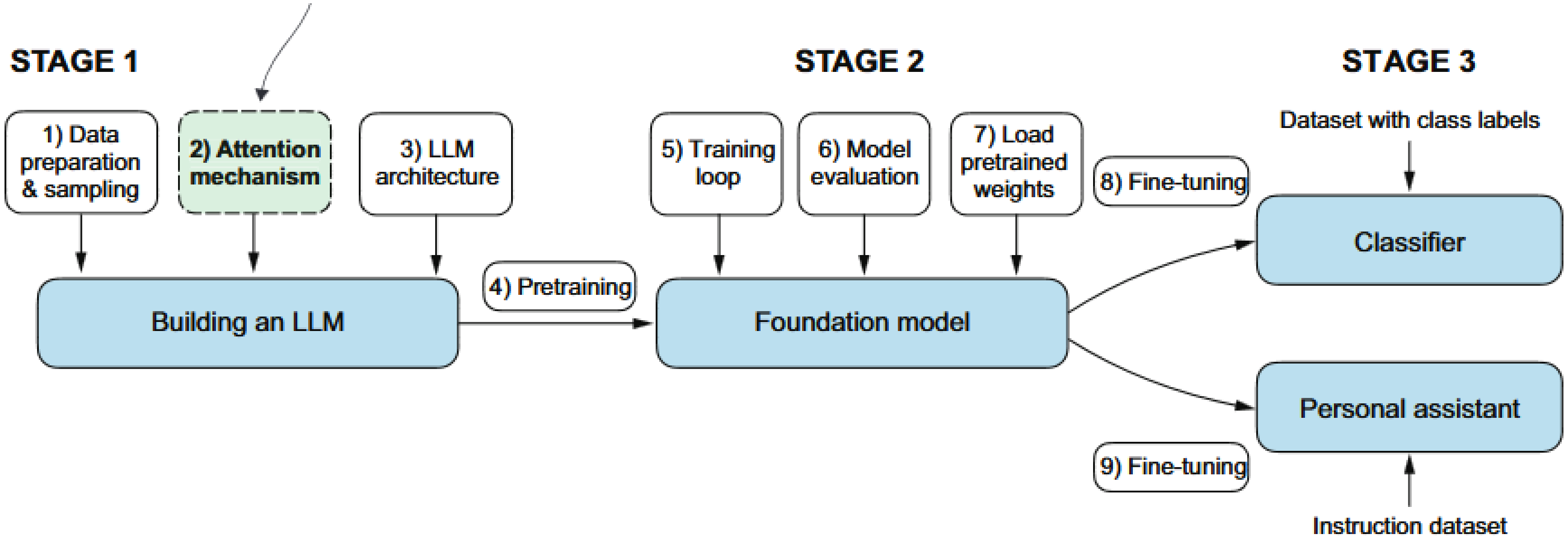
$$\theta_i = 10000^{-2(i-1)/d}$$

Encoding word positions

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

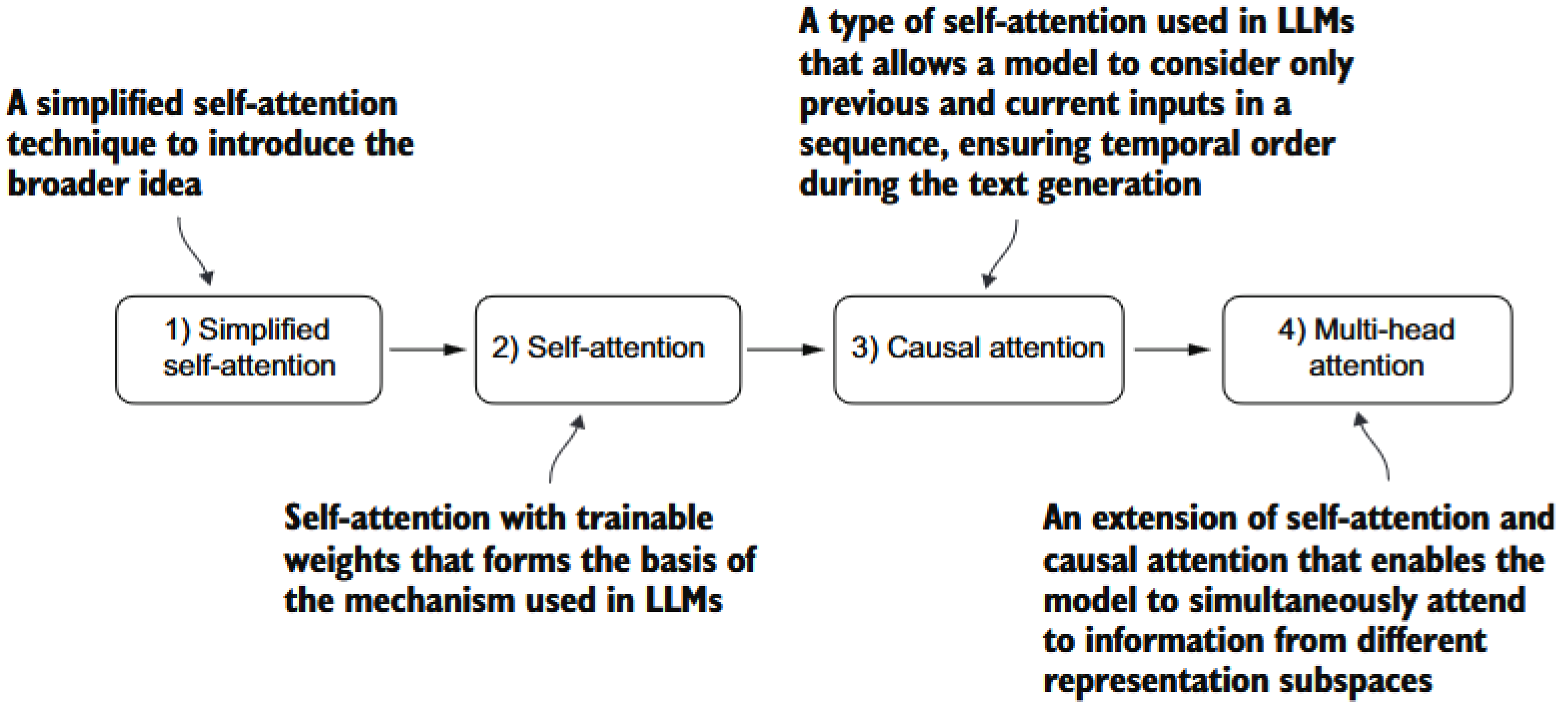


Attention mechanism

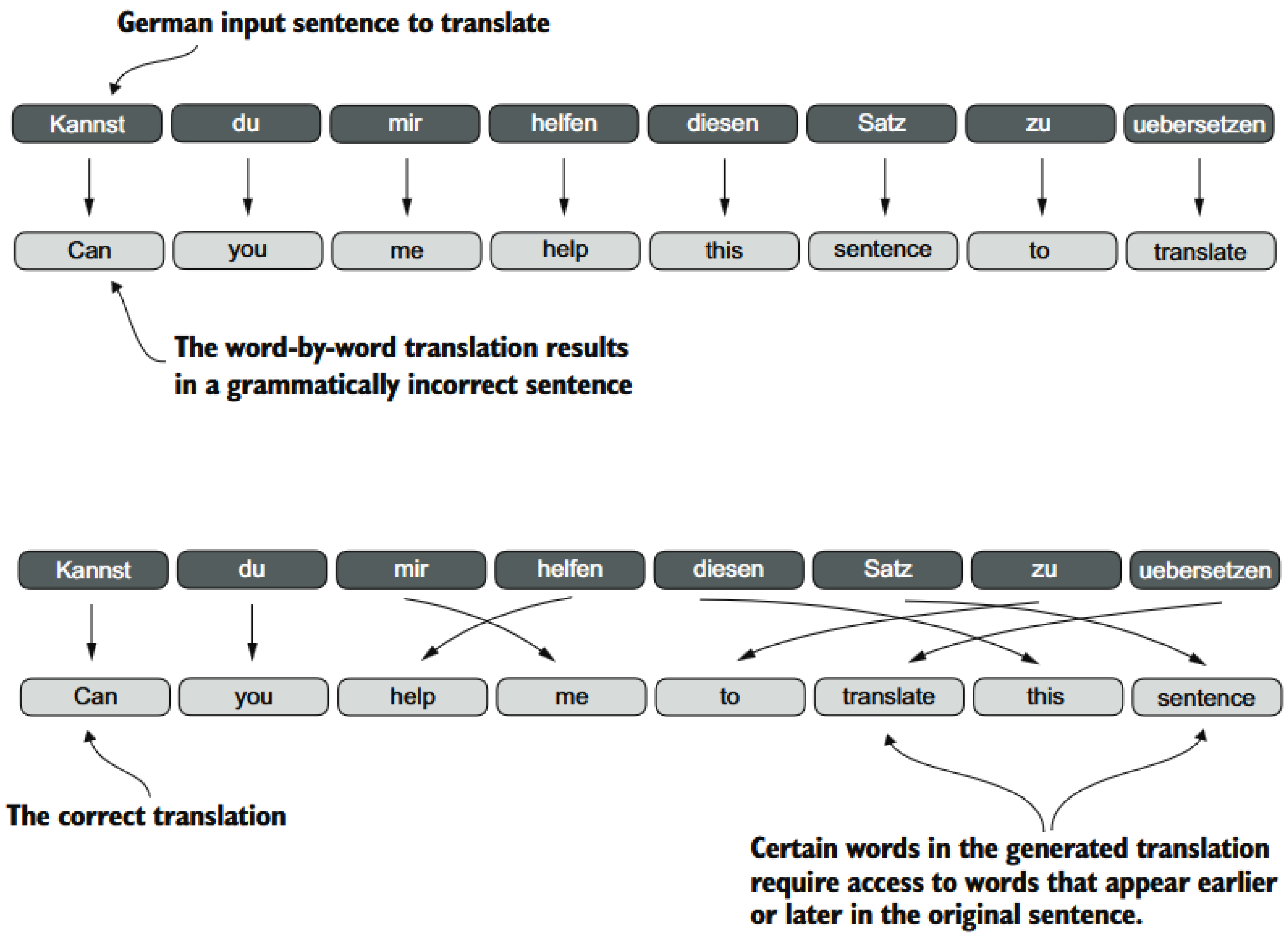


Attention mechanism

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería



Problem with modeling long sequences



Modeling sequences

Escuela de
Ciencias Aplicadas
e Ingeniería

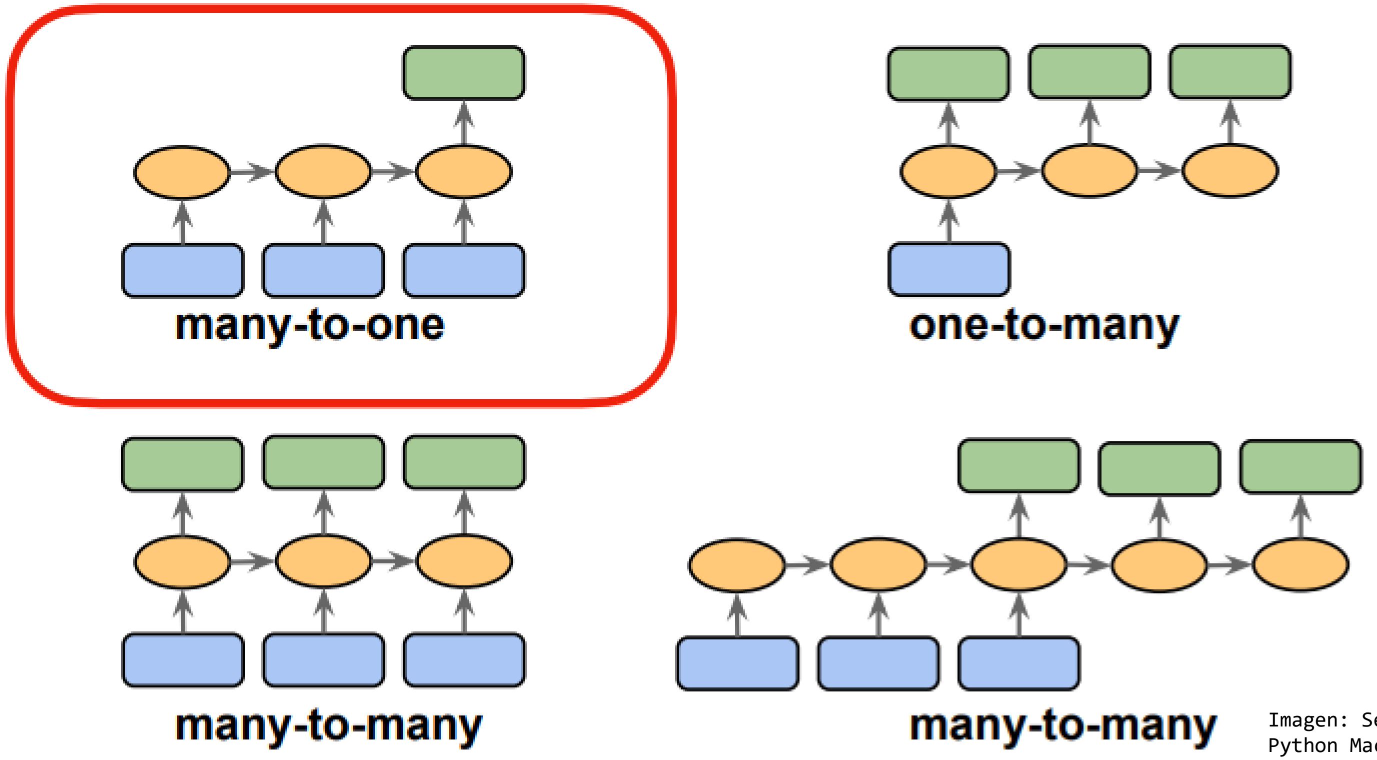
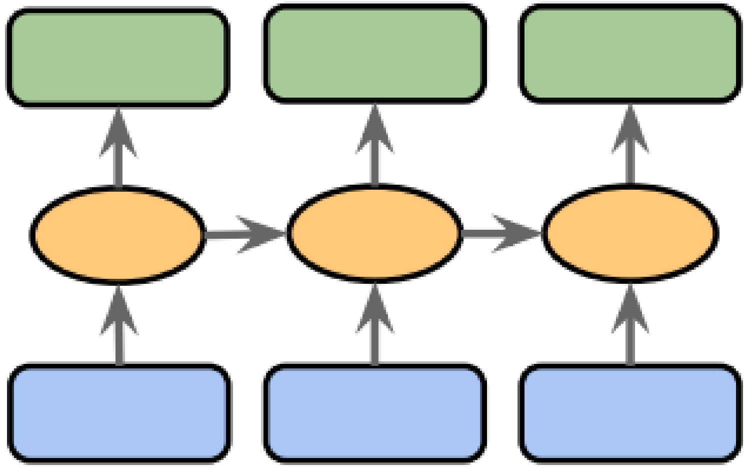


Imagen: Sebastian Raschka, Vahid Mirjalili.
Python Machine Learning. 3rd Edition.
Birmingham, UK: Packt Publishing, 2019

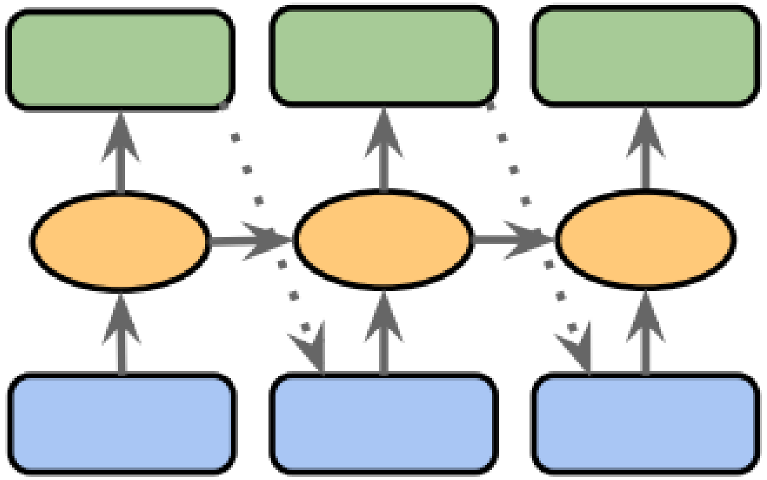
The Unreasonable Effectiveness of Recurrent Neural Networks by Andrej Karpathy (http://karpathy.github.io/2015/05/21/rnn-effectiveness/)

Modeling sequences



many-to-many

"training"

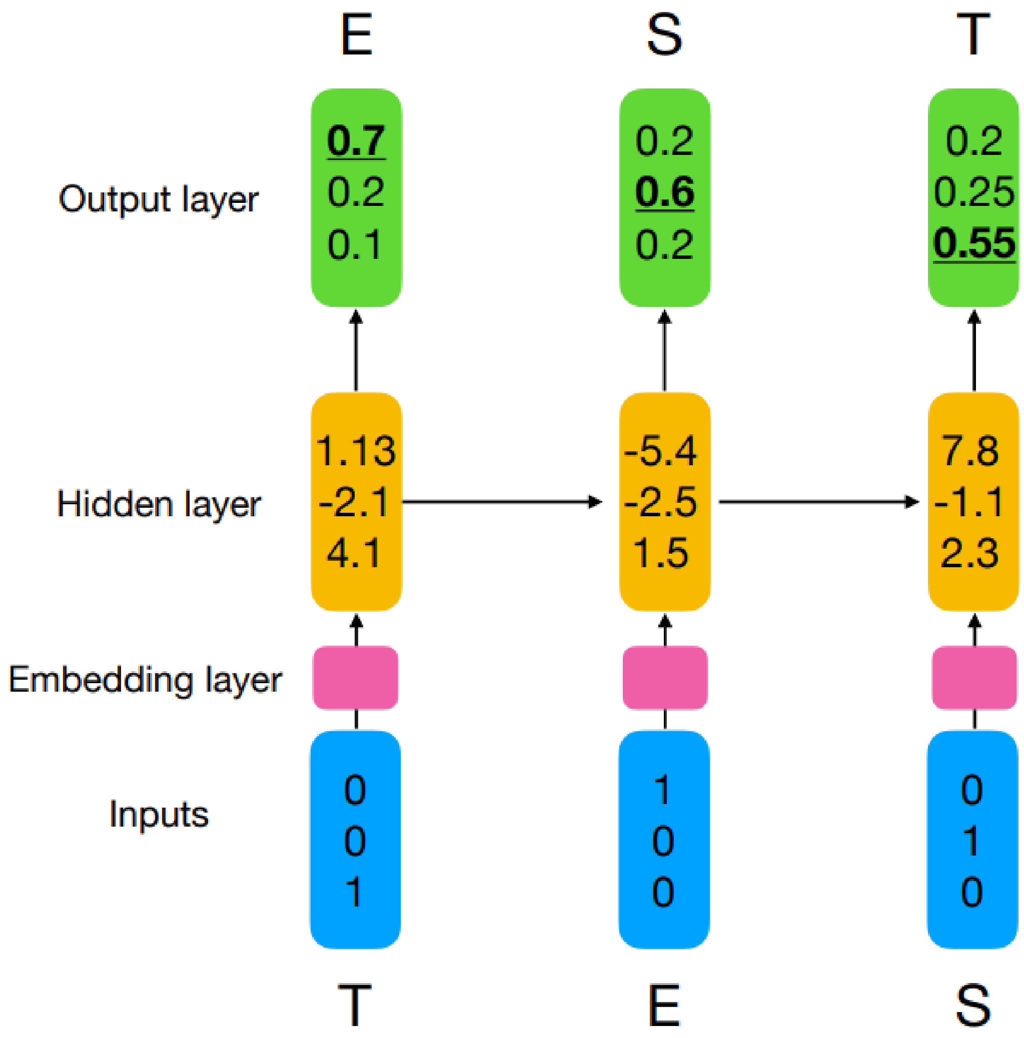


~~many~~-to-many

"one"

"generating new text"

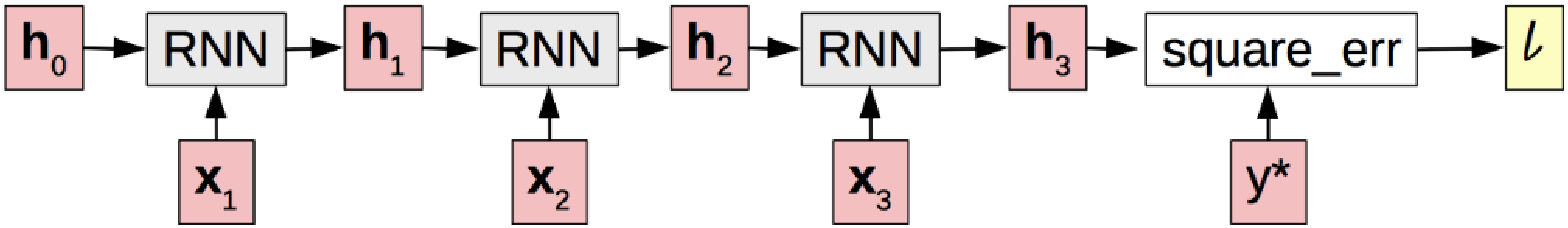
Recurrent Neural Networks



Recurrent Neural Networks

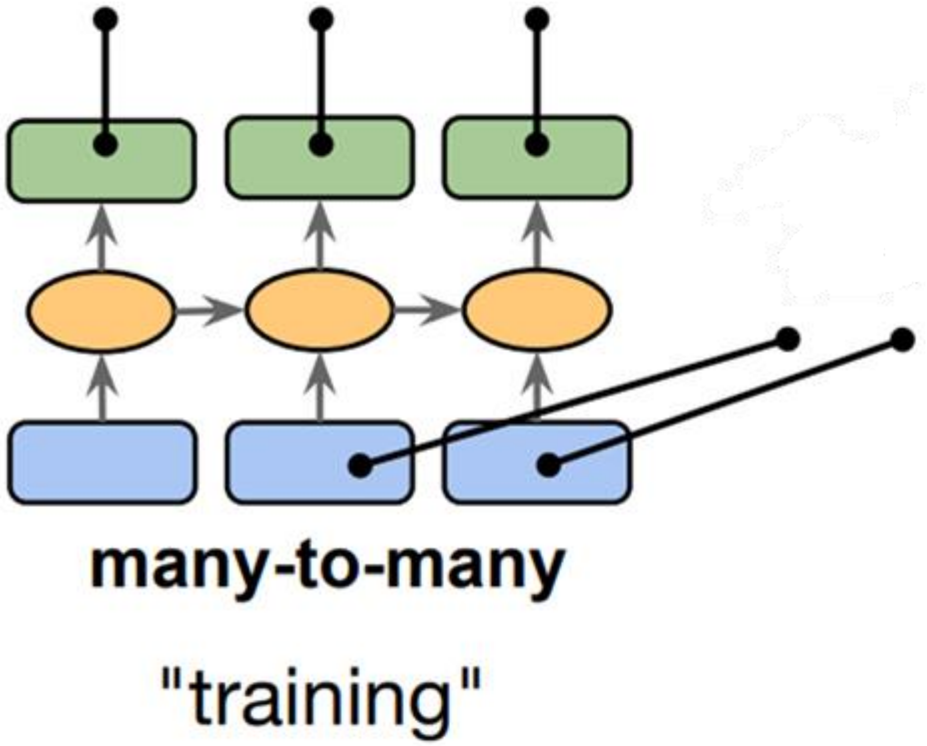
In RNNs, we asked about how backdrop through a network causes gradients can vanish or explode

$$\frac{dl}{d_{h_0}} = \text{tiny} \qquad \frac{dl}{d_{h_1}} = \text{small} \qquad \frac{dl}{d_{h_2}} = \text{med.} \qquad \frac{dl}{d_{h_3}} = \text{large}$$



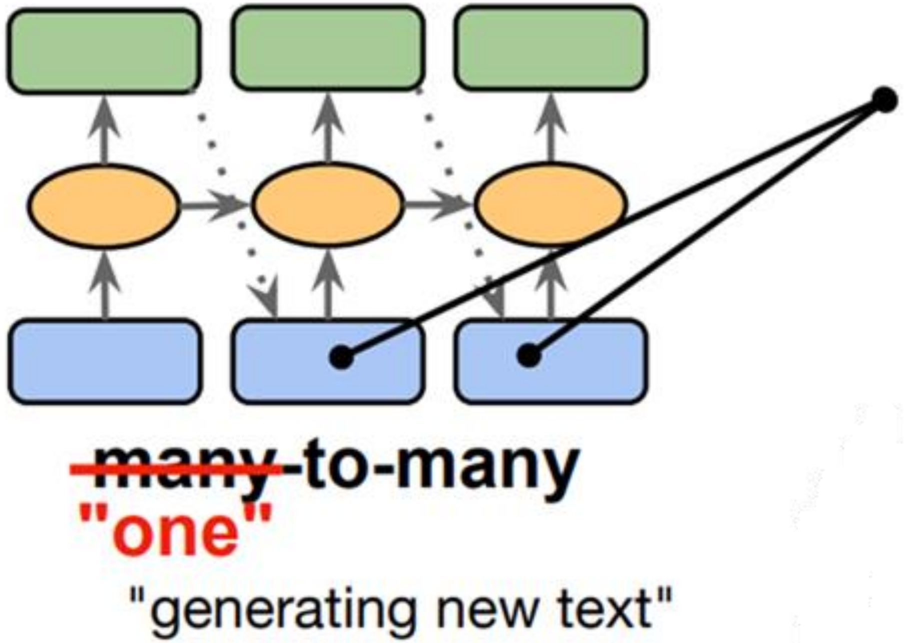
Recurrent Neural Networks

At each time step, Softmax output (probability) for each possible 'next letter



For the next input, ignore the prediction but use the 'correct' next letter from the dataset.

Recurrent Neural Networks



To generate new text, now display the softmax outputs and provide the letter as input for the next time step.

Long Short Term Memory Networks

UNIVERSIDAD EAFIT

Escuela de Ciencias Aplicadas e Ingeniería

https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html

Parameters

- **input_size** – The number of expected features in the input *x*
- **hidden_size** – The number of features in the hidden state *h*
- **num_layers** – Number of recurrent layers. E.g., setting `num_layers=2` would mean stacking two LSTMs together to form a *stacked LSTM*, with the second LSTM taking in outputs of the first LSTM and computing the final results. Default: 1
- **bias** – If `False`, then the layer does not use bias weights *b_ih* and *b_hh*. Default: `True`
- **batch_first** – If `True`, then the input and output tensors are provided as (batch, seq, feature). Default: `False`
- **dropout** – If non-zero, introduces a *Dropout* layer on the outputs of each LSTM layer except the last layer, with dropout probability equal to `dropout`. Default: 0
- **bidirectional** – If `True`, becomes a bidirectional LSTM. Default: `False`
- **proj_size** – If `> 0`, will use LSTM with projections of corresponding size. Default: 0

Examples:

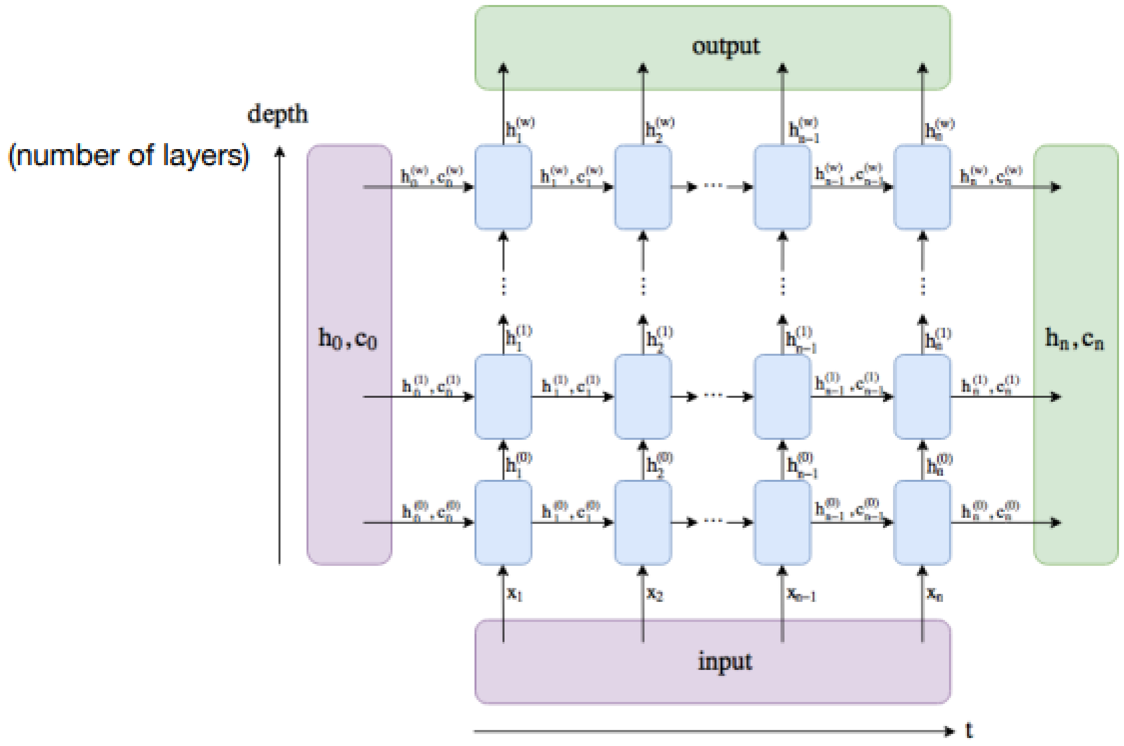
```
>>> rnn = nn.LSTM(10, 20, 2)
>>> input = torch.randn(5, 3, 10)
>>> h0 = torch.randn(2, 3, 20)
>>> c0 = torch.randn(2, 3, 20)
>>> output, (hn, cn) = rnn(input, (h0, c0))
```

LSTM

Universidad EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

Examples:

```
>>> rnn = nn.LSTM(10, 20, 2)
>>> input = torch.randn(5, 3, 10)
>>> h0 = torch.randn(2, 3, 20)
>>> c0 = torch.randn(2, 3, 20)
>>> output, (hn, cn) = rnn(input, (h0, c0))
```



Fuente de imagen: https://stackoverflow.com/questions/48302810/whats-the-difference-between-hidden-and-output-in-pytorch-lstm

LSTM

https://pytorch.org/docs/stable/generated/torch.nn.LSTMCell.html

Inputs: input, (h_0, c_0)

- **input** of shape (*batch, input_size*): tensor containing input features
- **h_0** of shape (*batch, hidden_size*): tensor containing the initial hidden state for each element in the batch.
- **c_0** of shape (*batch, hidden_size*): tensor containing the initial cell state for each element in the batch.

  If (*h_0, c_0*) is not provided, both **h_0** and **c_0** default to zero.

Outputs: (h_1, c_1)

- **h_1** of shape (*batch, hidden_size*): tensor containing the next hidden state for each element in the batch
- **c_1** of shape (*batch, hidden_size*): tensor containing the next cell state for each element in the batch

Examples:

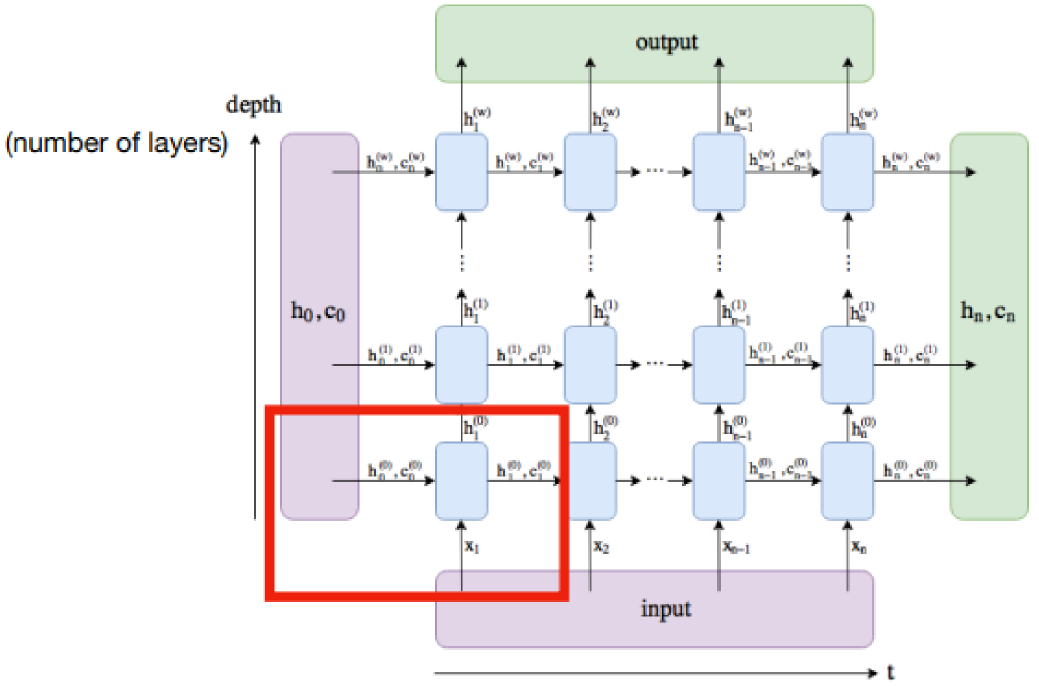
```
>>> rnn = nn.LSTMCell(10, 20) # (input_size, hidden_size)
>>> input = torch.randn(2, 3, 10) # (time_steps, batch, input_size)
>>> hx = torch.randn(3, 20) # (batch, hidden_size)
>>> cx = torch.randn(3, 20)
>>> output = []
>>> for i in range(input.size()[0]):
        hx, cx = rnn(input[i], (hx, cx))
        output.append(hx)
>>> output = torch.stack(output, dim=0)
```

Clase LSTM

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

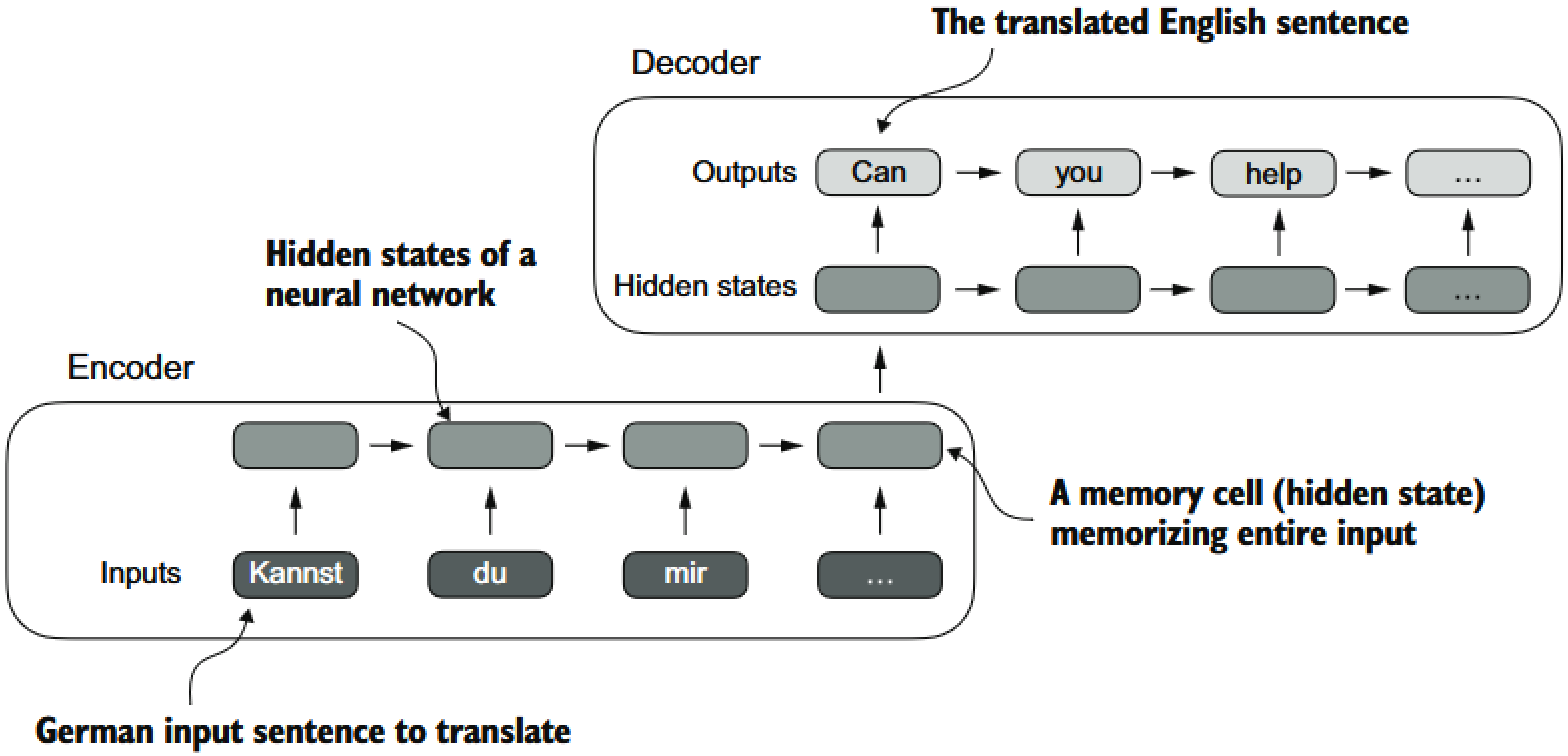
Examples:

```
>>> rnn = nn.LSTMCell(10, 20) # (input_size, hidden_size)
>>> input = torch.randn(2, 3, 10) # (time_steps, batch, input_size)
>>> hx = torch.randn(3, 20) # (batch, hidden_size)
>>> cx = torch.randn(3, 20)
>>> output = []
>>> for i in range(input.size()[0]):
        hx, cx = rnn(input[i], (hx, cx))
        output.append(hx)
>>> output = torch.stack(output, dim=0)
```



Fuente de imagen: https://stackoverflow.com/questions/48302810/whats-the-difference-between-hidden-and-output-in-pytorch-lstm

Seq2seq



The translated English sentence
Decoder
Outputs
Can
you
help
...
Hidden states
...
Hidden states of a neural network
Encoder
A memory cell (hidden state) memorizing entire input
Inputs
Kannst
du
mir
...
German input sentence to translate

Attention mechanism

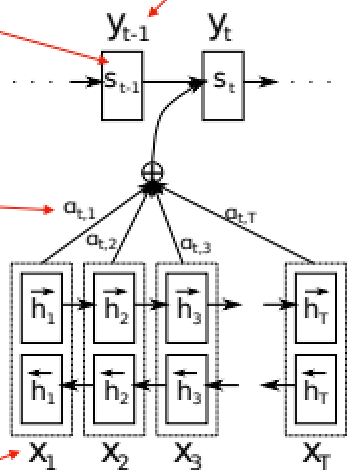
Assign an attention weight to each word to determine how much 'attention' the model should pay to each word (that is, for each word, the network learns a 'context').

Attention mechanism

Hidden state in a regular RNN
(RNN #2)

1st translated word

Attention weight



Bidirectional RNN
(RNN #1)

1st input word

Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.

NEURAL MACHINE TRANSLATION
BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

Dzmitry Bahdanau
Jacobs University Bremen, Germany

KyungHyun Cho **Yoshua Bengio***
Université de Montréal

ABSTRACT

Attention is all you need

Escuela de Ciencias Aplicadas e Ingeniería

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

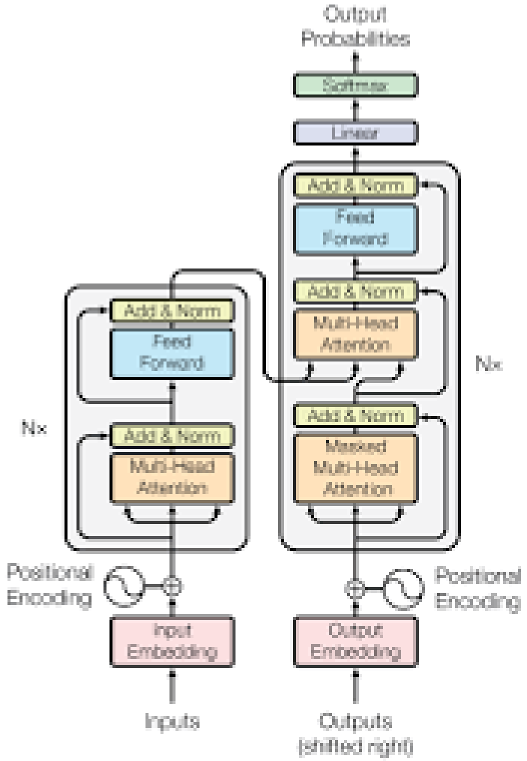
Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

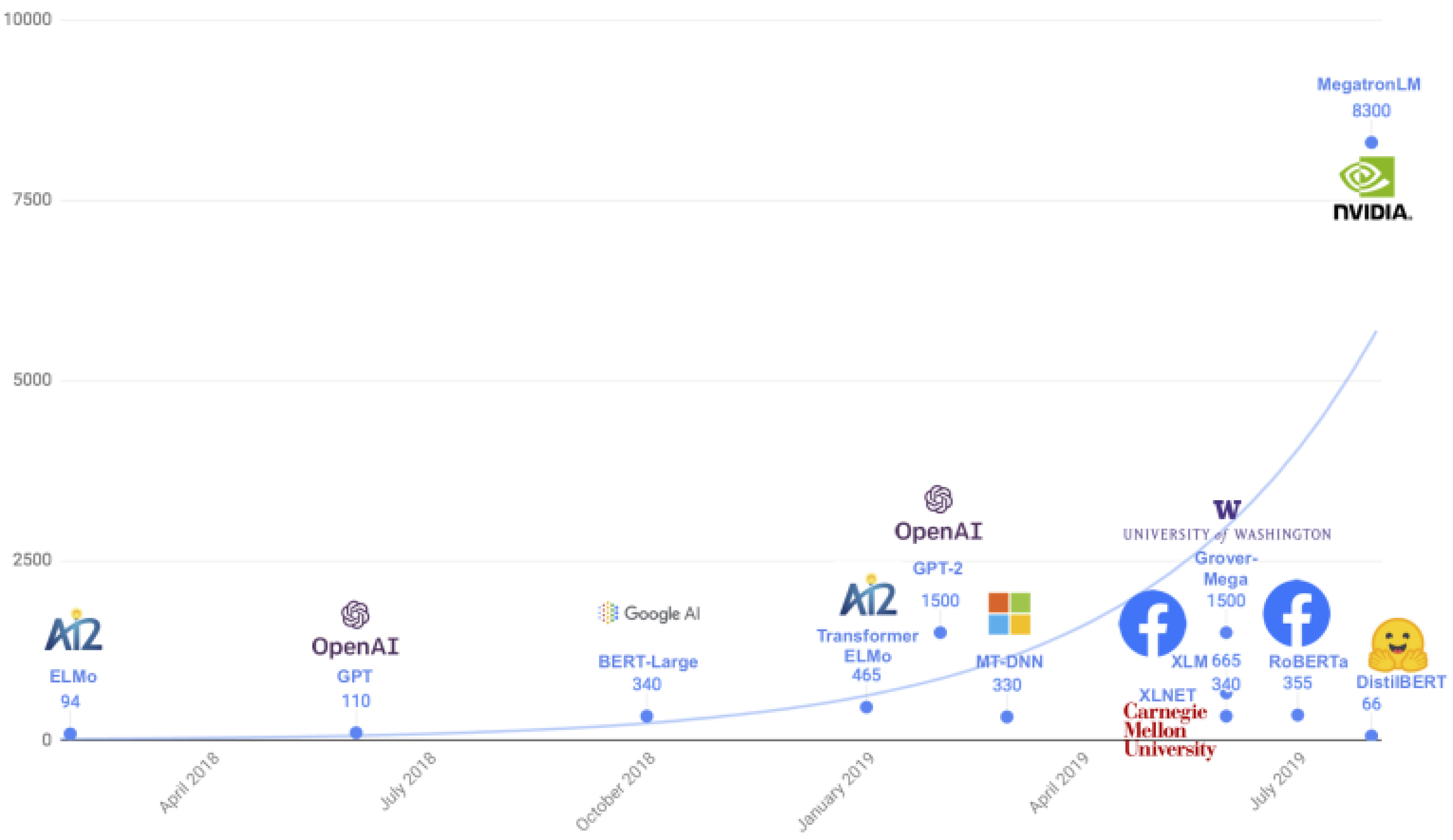
Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com



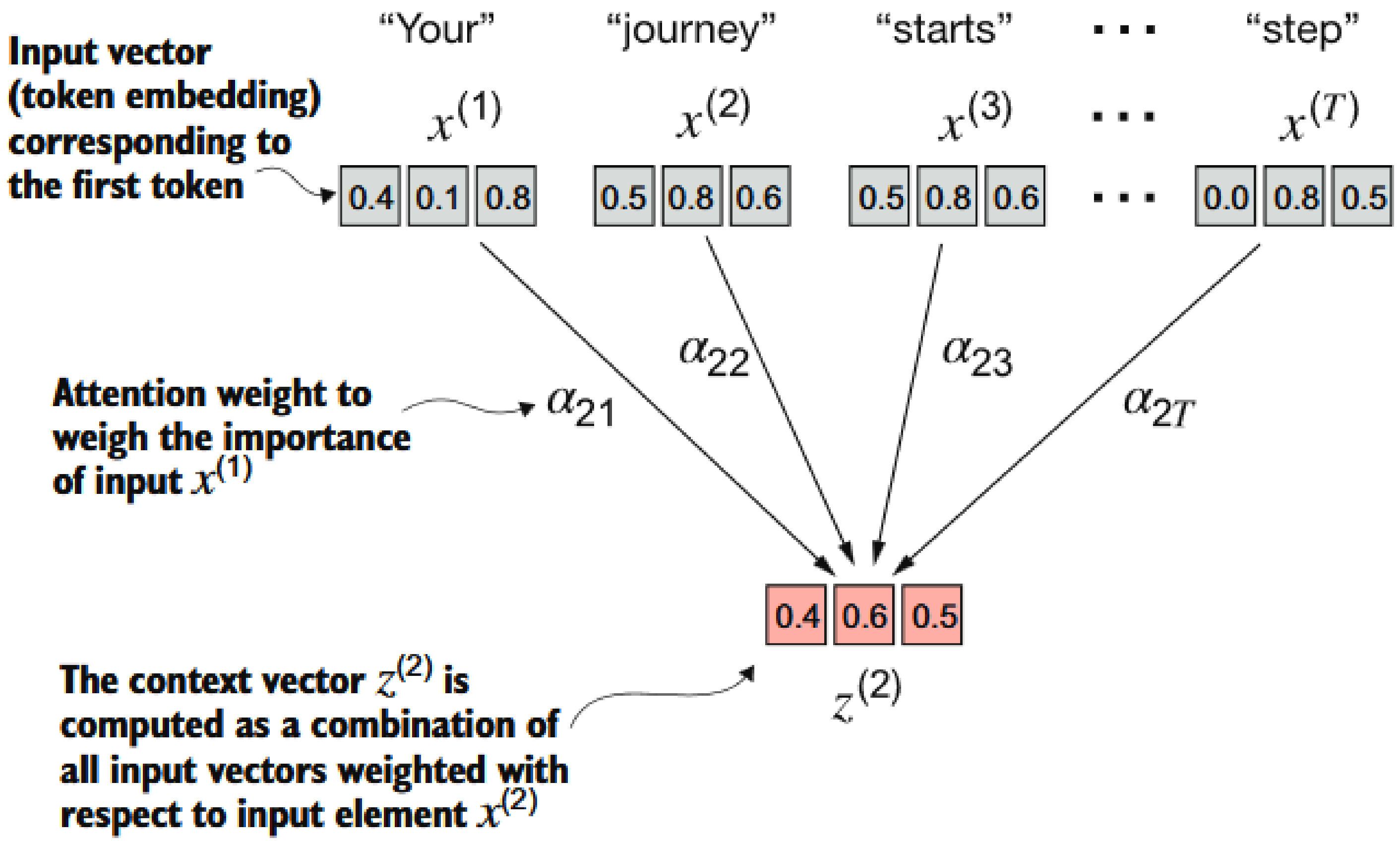
Since ~2018, transformers have been growing in popularity... and size

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería



Fuente imagen: https://medium.com/huggingface/distilbert-8cf3380435b5

A simple self-attention mechanism without trainable weights



A simple self-attention mechanism without trainable weights

Universidad EAFIT | Escuela de Ciencias Aplicadas e Ingeniería

Self-attention as weighted sum:

$$\boldsymbol{A}_i = \sum_{j=0}^{T} a_{ij}\boldsymbol{x}_j$$

output corresponding to the i-th input

weight based on similarity between current input $\boldsymbol{x}_i$ and all other inputs

How to compute the attention weights?

here as simple dot product:

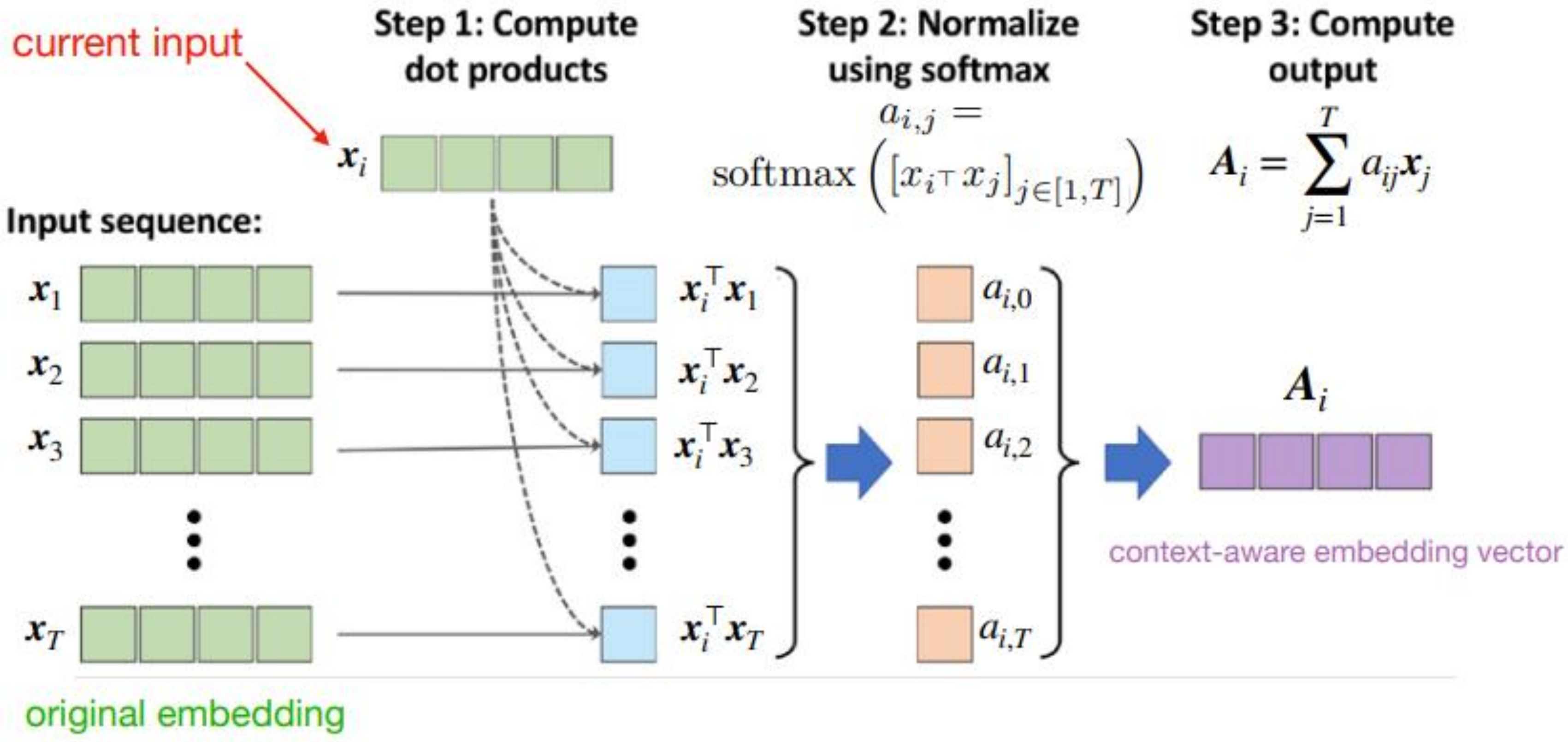
$$e_{ij} = \boldsymbol{x}_i^{\top}\boldsymbol{x}_j$$

repeat this for all inputs $j \in \{1 \dots T\}$, then normalize

$$a_{ij} = \frac{\exp\left(e_{ij}\right)}{\sum_{j=1}^{T}\exp\left(e_{ij}\right)} = \operatorname{softmax}\left(\left[e_{ij}\right]_{j=1\dots T}\right)$$

A simple self-attention mechanism without trainable weights

Self-attention: Relating different positions within a single sequence (vs. between input and output sequences



Raschka & Mirjalili 2019. Python Machine Learning, 3rd edition

Self-attention mechanism

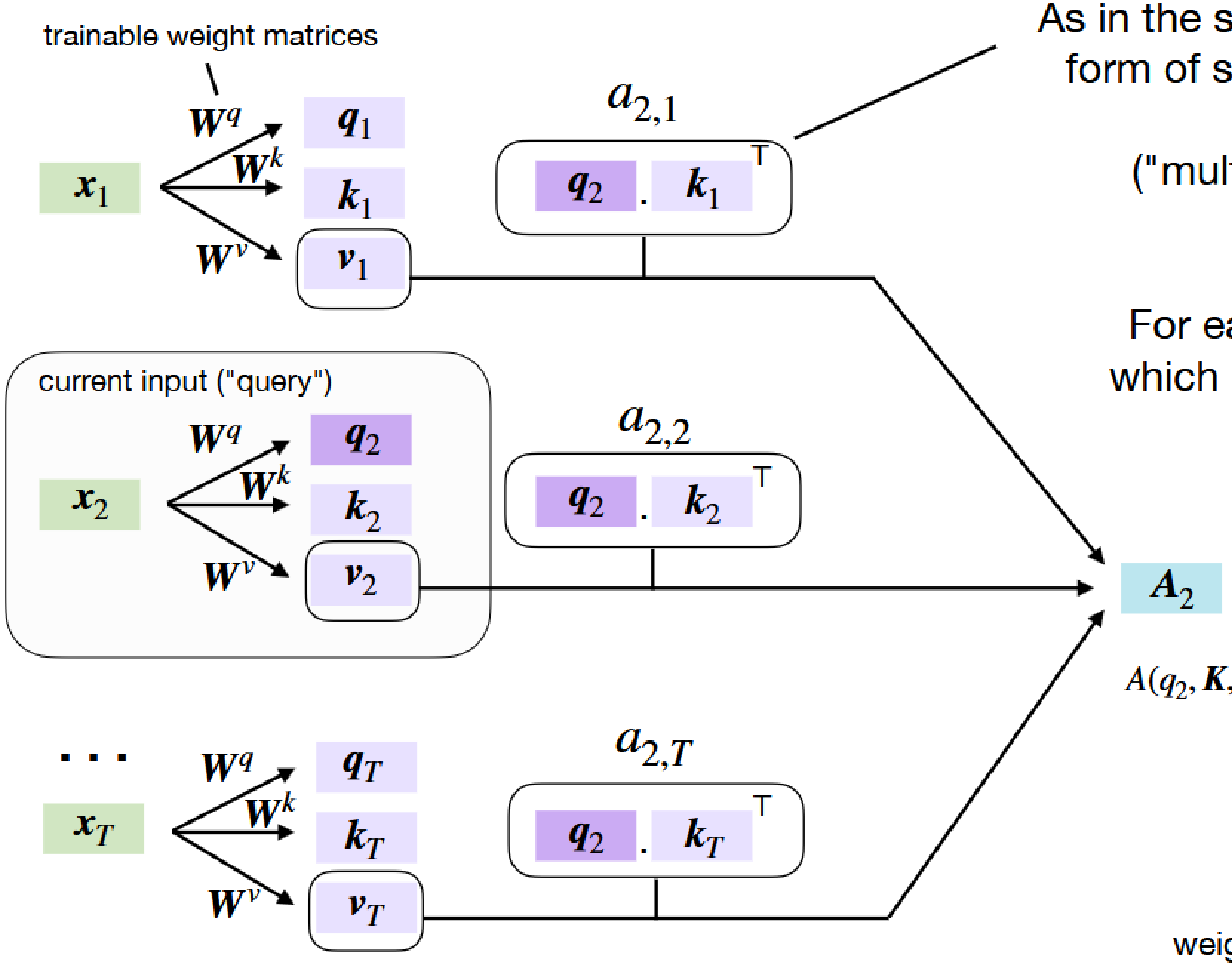
- Previous basic version did not involve any learnable parameters, so not very useful for learning a language model

- We are now adding 3 trainable weight matrices that are multiplied with the input sequence embeddings

$$\text{query} = \boldsymbol{W}^{q}\boldsymbol{x}_i$$
$$\text{key} = \boldsymbol{W}^{k}\boldsymbol{x}_i$$
$$\text{value} = \boldsymbol{W}^{v}\boldsymbol{x}_i$$

Self-attention mechanism

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería



As in the simplified version, this is a form of similarity or compatibility measure ("multiplicative attention")

For each query, model learns which **k**ey-**v**alue input it should attend to

$$A(q_2, \boldsymbol{K}, \boldsymbol{V}) = \sum_{i=1}^{T} \Big[\underbrace{\frac{\exp(q_2 \cdot k_i^\top)}{\sum_j \exp(q_2 \cdot k_j^\top)}}_{\text{softmax}} \times \boldsymbol{v}_i \Big]$$

weighted sum: values weighted by attention weight (softmax score)

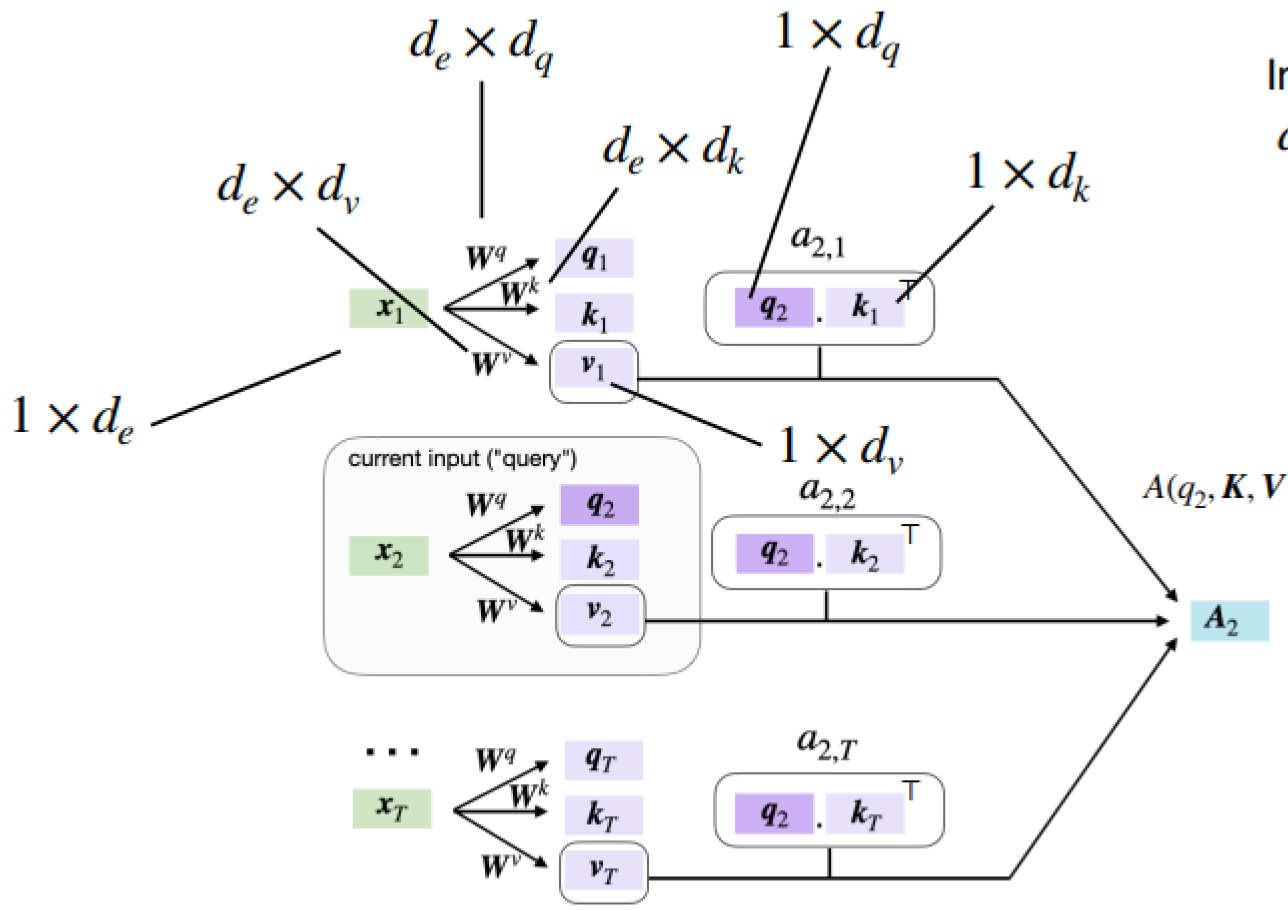
Self-attention mechanism

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

d_e = embedding size (original transformer = 512)

where $d_q = d_k$

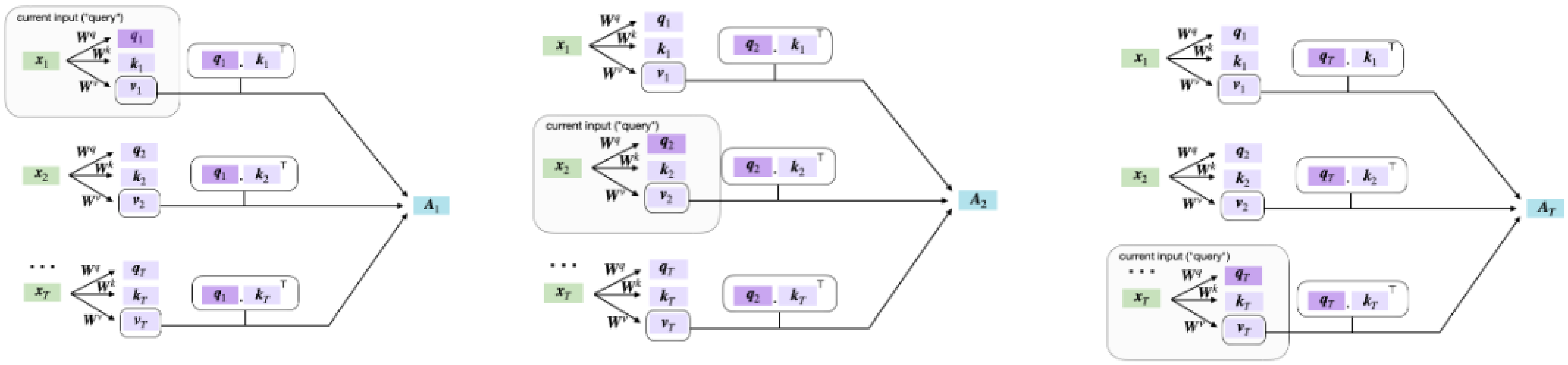
In original transformer, $d_q = d_v$ as well



$$A(q_2, \boldsymbol{K}, \boldsymbol{V}) = \sum_{i=1}^{T} \Big[\underbrace{\frac{\exp(q_2 \cdot k_i^\top)}{\sum_j \exp(q_2 \cdot k_j^\top)}}_{\substack{\text{softmax} \\ 1 \times 1}} \times \boldsymbol{v}_i\Big]$$

$1 \times d_v$

Self-attention mechanism



Attention score matrix: $$A = \begin{bmatrix} A_1 \\ A_2 \\ A_3 \end{bmatrix}$$

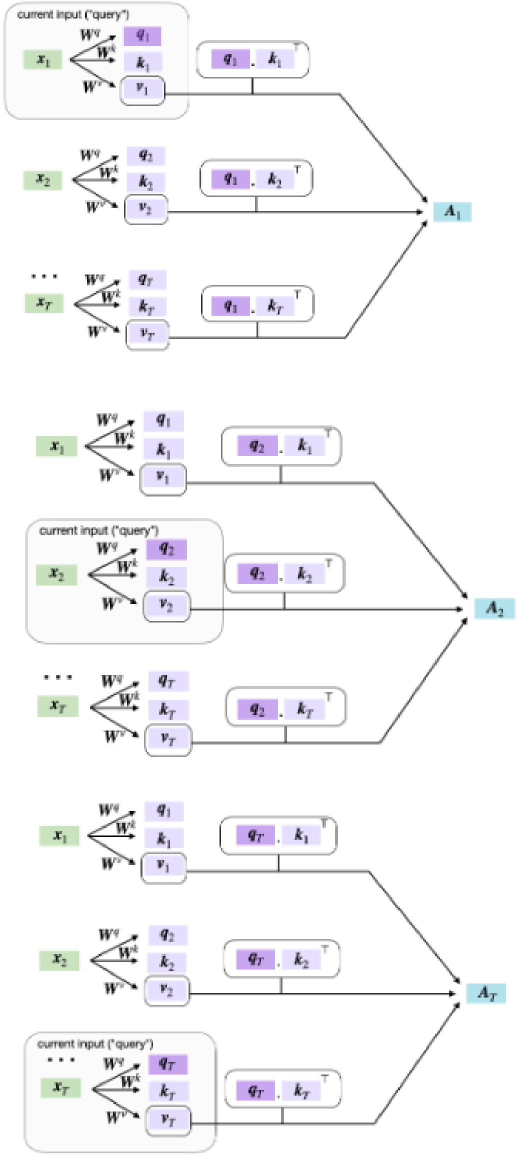
Self attention mechanism – Scaled dot producto attention

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

d_e = embedding size

T = input sequence size

$$\boldsymbol{x} \in \mathbb{R}^{T \times d_e}$$



$$\boldsymbol{Q} \in \mathbb{R}^{T \times d_q}$$

$$\boldsymbol{K} \in \mathbb{R}^{T \times d_k}$$

$$\boldsymbol{V} \in \mathbb{R}^{T \times d_v}$$

"attention matrix"

$$A(Q,K,V) = \underbrace{\overbrace{\text{softmax}\left(\frac{QK^{\top}}{\sqrt{d_k}}\right)}^{T \times T} V}_{T \times d_v}$$

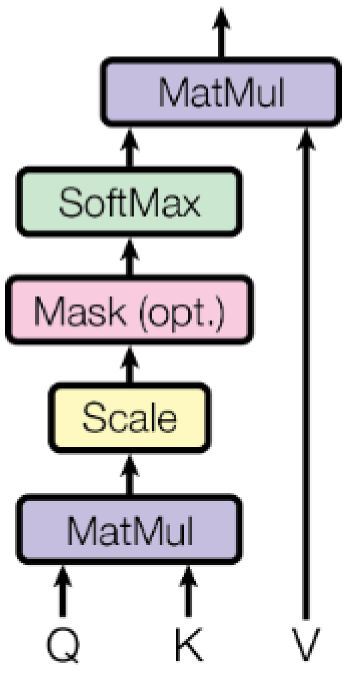
"attention-based embedding"

Scaled dot product attention

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

To ensure that the dot-products between query and and key don't grow too large (and softmax gradient become too small) for large d_k

Scaled Dot-Product Attention



Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention Is All You Need.

Multi-Head attention

- Apply self-attention multiple times in parallel (similar to multiple kernels for channels in CNNs)

- For each head (self-attention layer), use different $\mathbf{W}^q, \mathbf{W}^k, \mathbf{W}^v$ then concatenate the results $\mathbf{A}_{(i)}$.

- 8 attention heads in the original transformer.

- Allows attending to different parts in the sequence differently.

Multi-Head attention

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería

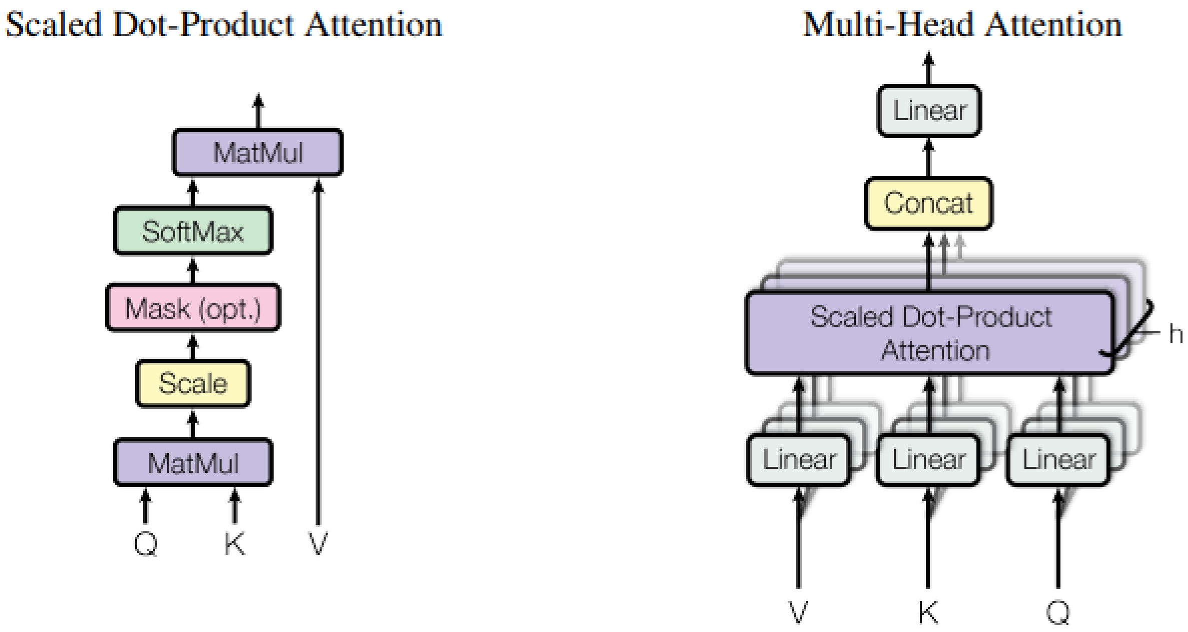
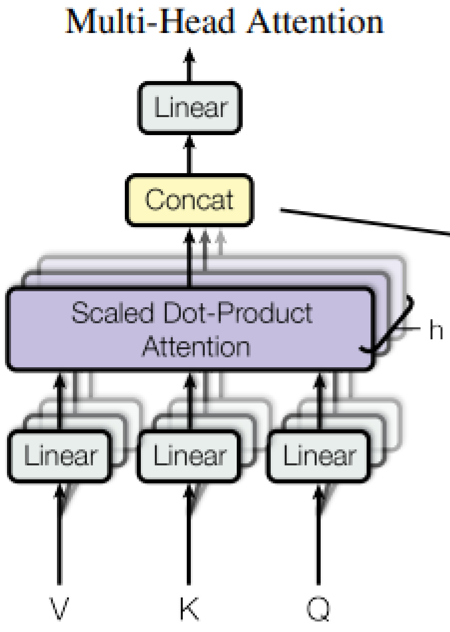


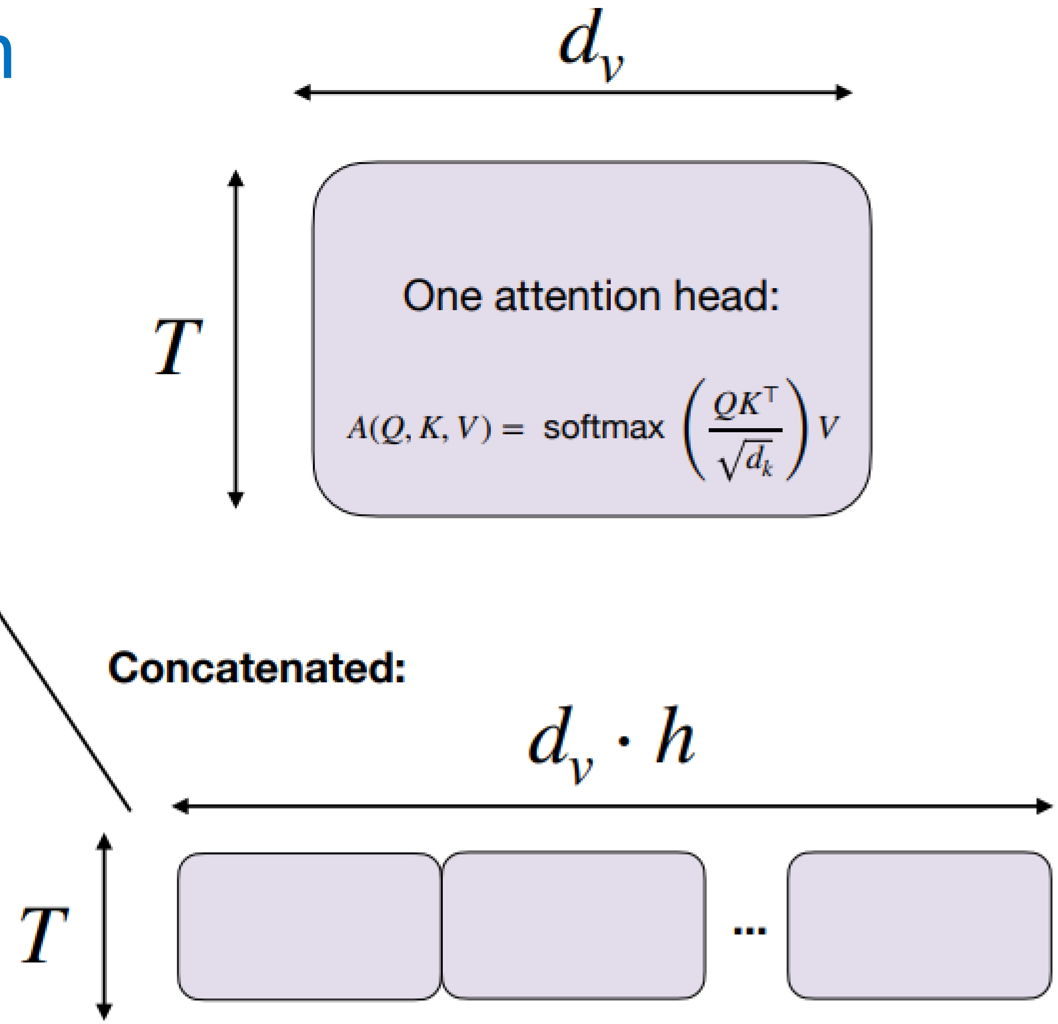
Figure 2: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention Is All You Need.

Multi-Head attention

Escuela de
Ciencias Aplicadas e Ingeniería





Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention Is All You Need.

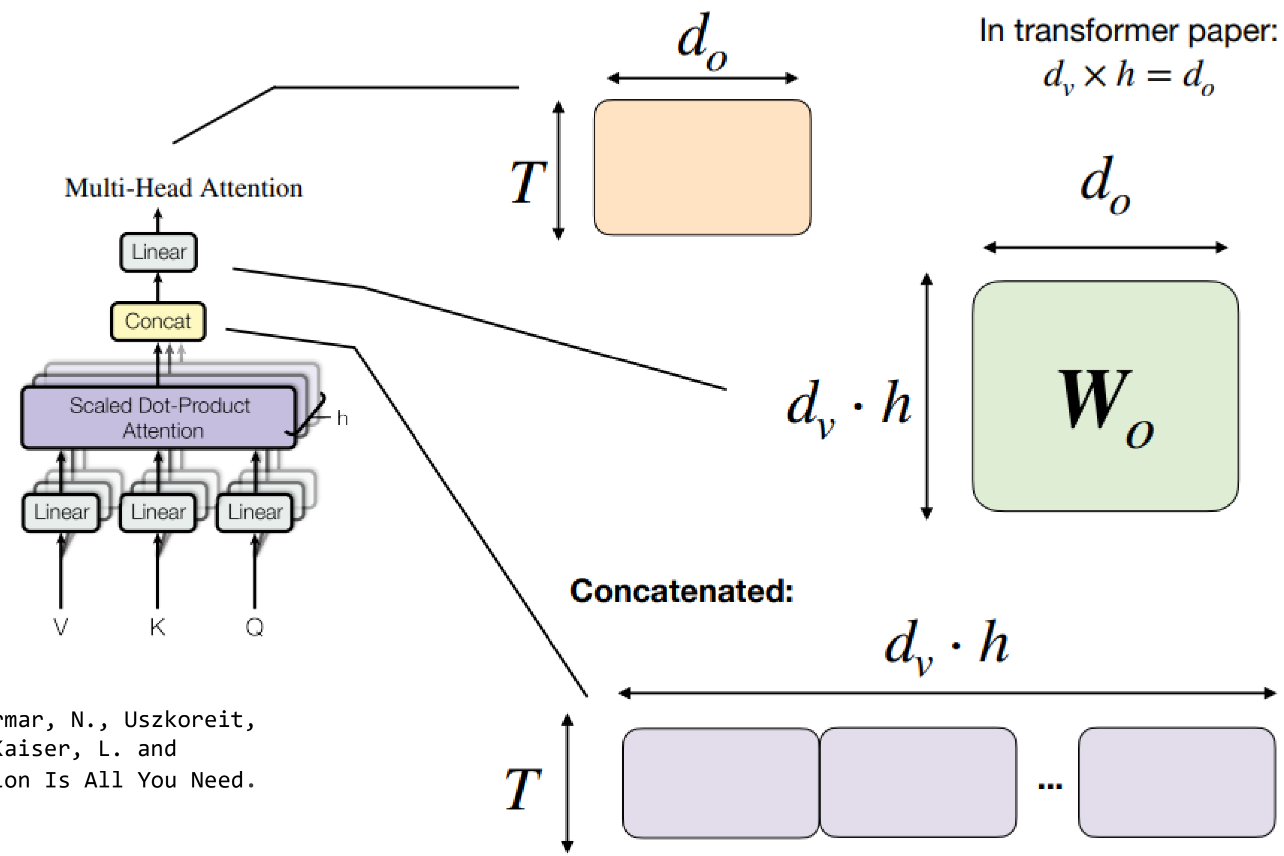
Dimensión de la sequencia de entrada en el transformador orginal:

$$T \times d_e = T \times 512$$

y

$$d_v = 512/h = 64$$

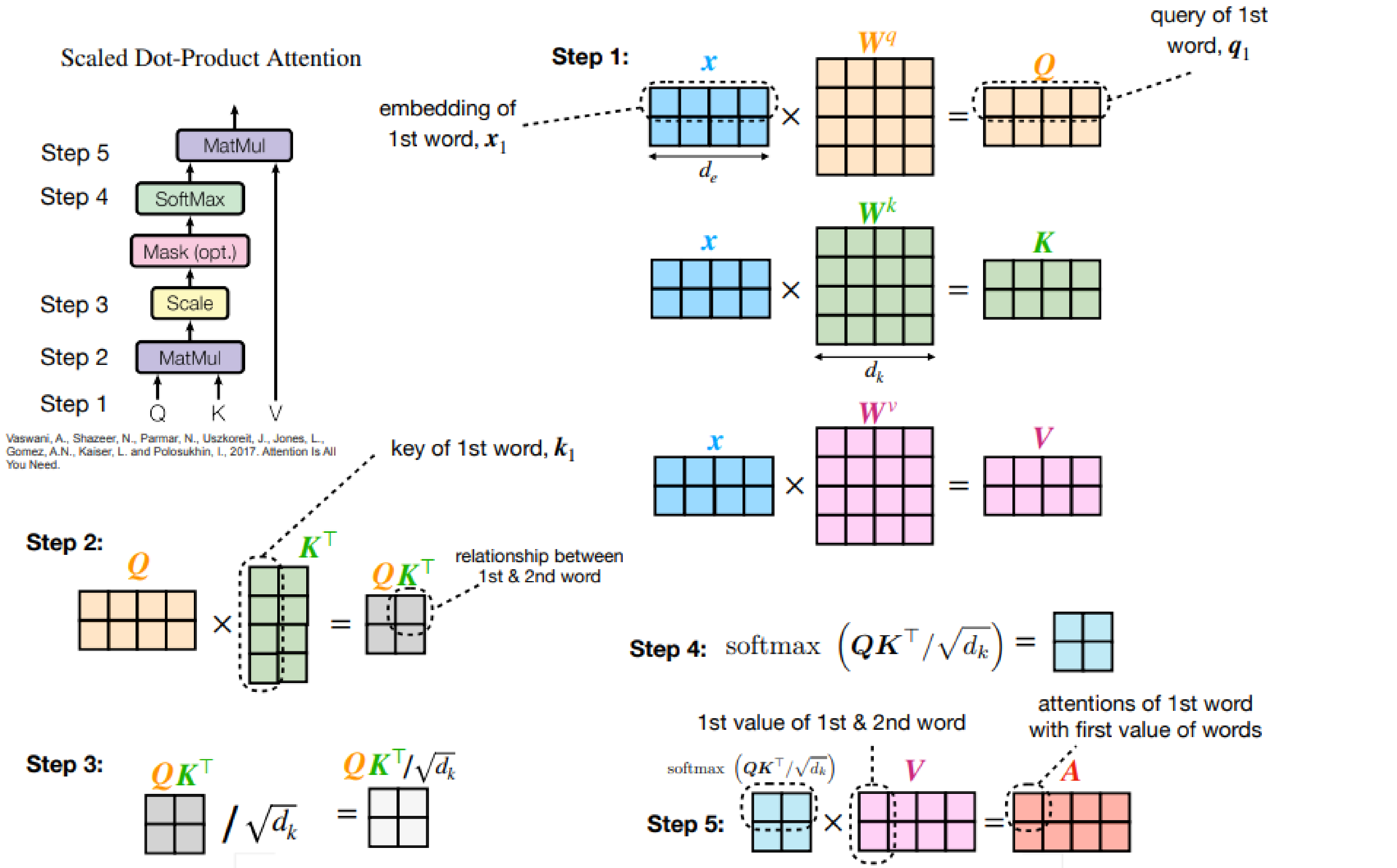
Multi-Head attention



Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention Is All You Need.

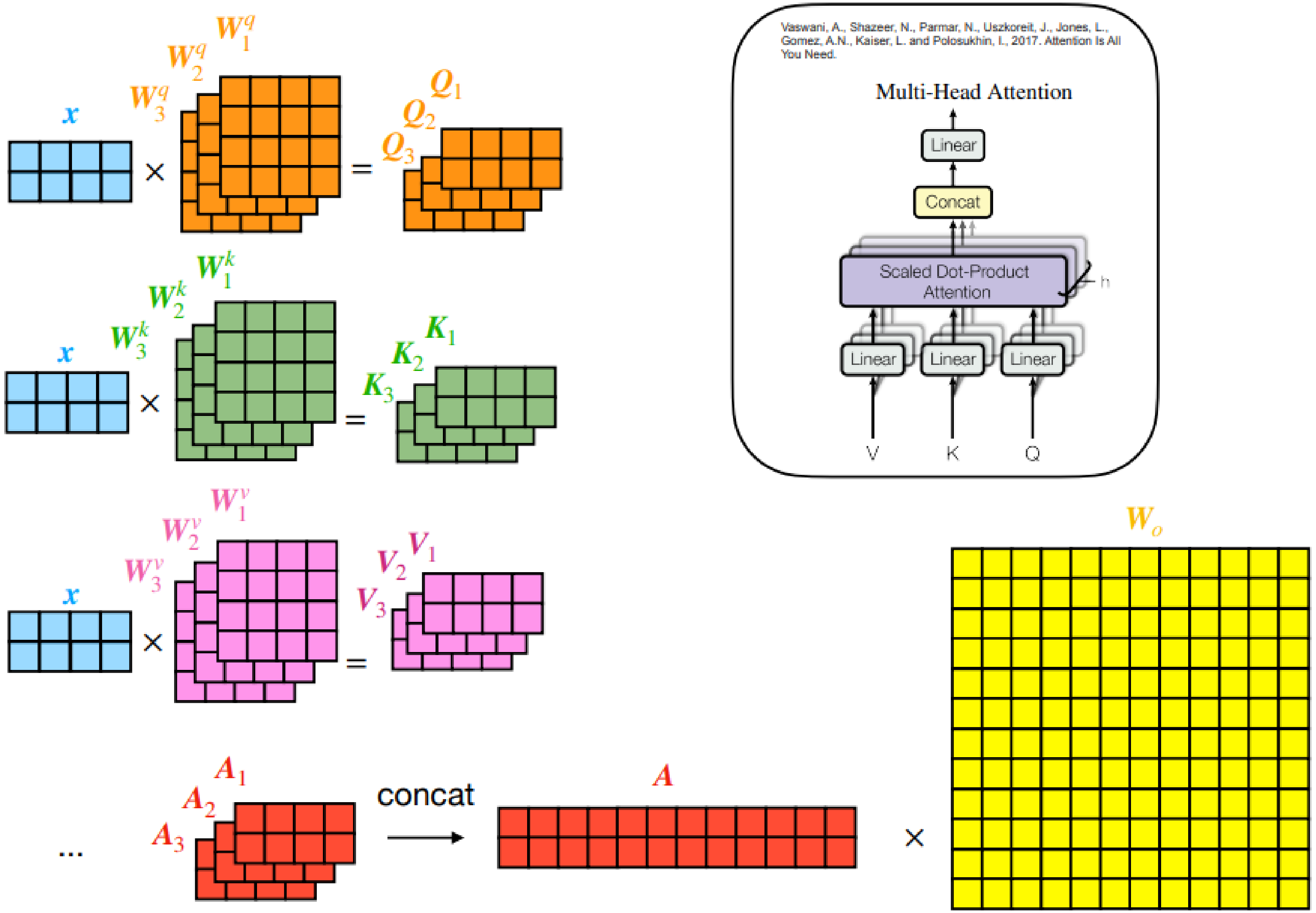
Scaled Dot-Product Attention Recap

UNIVERSIDAD EAFIT | Escuela de Ciencias Aplicadas e Ingeniería



Scaled Dot-Product Attention Recap

UNIVERSIDAD EAFIT
Escuela de Ciencias Aplicadas e Ingeniería



Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention Is All You Need.

Effects of Multi-Heads

- Example from Vaswani et al

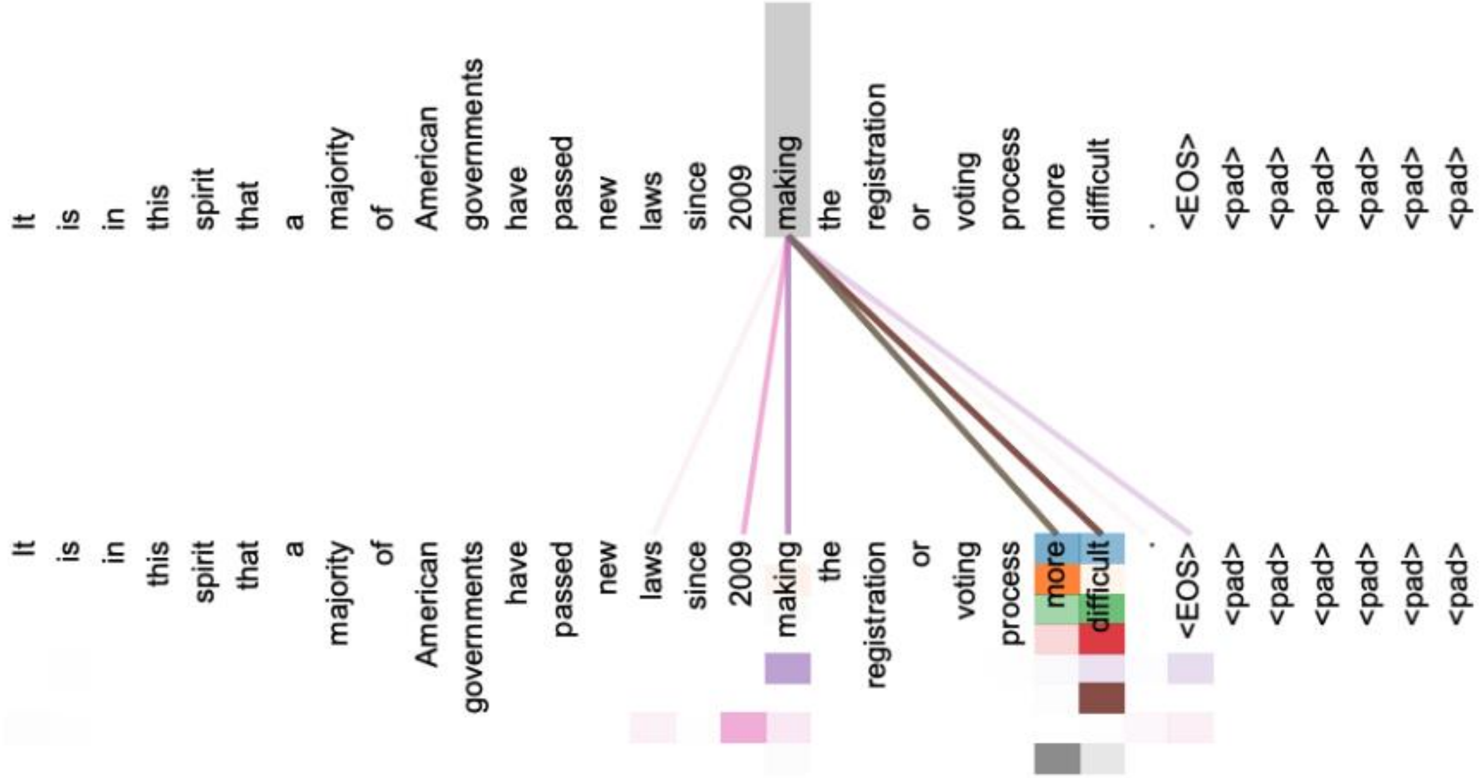


Figure 3: An example of the attention mechanism following long-distance dependencies in the encoder self-attention in layer 5 of 6. Many of the attention heads attend to a distant dependency of the verb ‘making’, completing the phrase ‘making...more difficult’. Attentions here shown only for the word ‘making’. Different colors represent different heads. Best viewed in color.

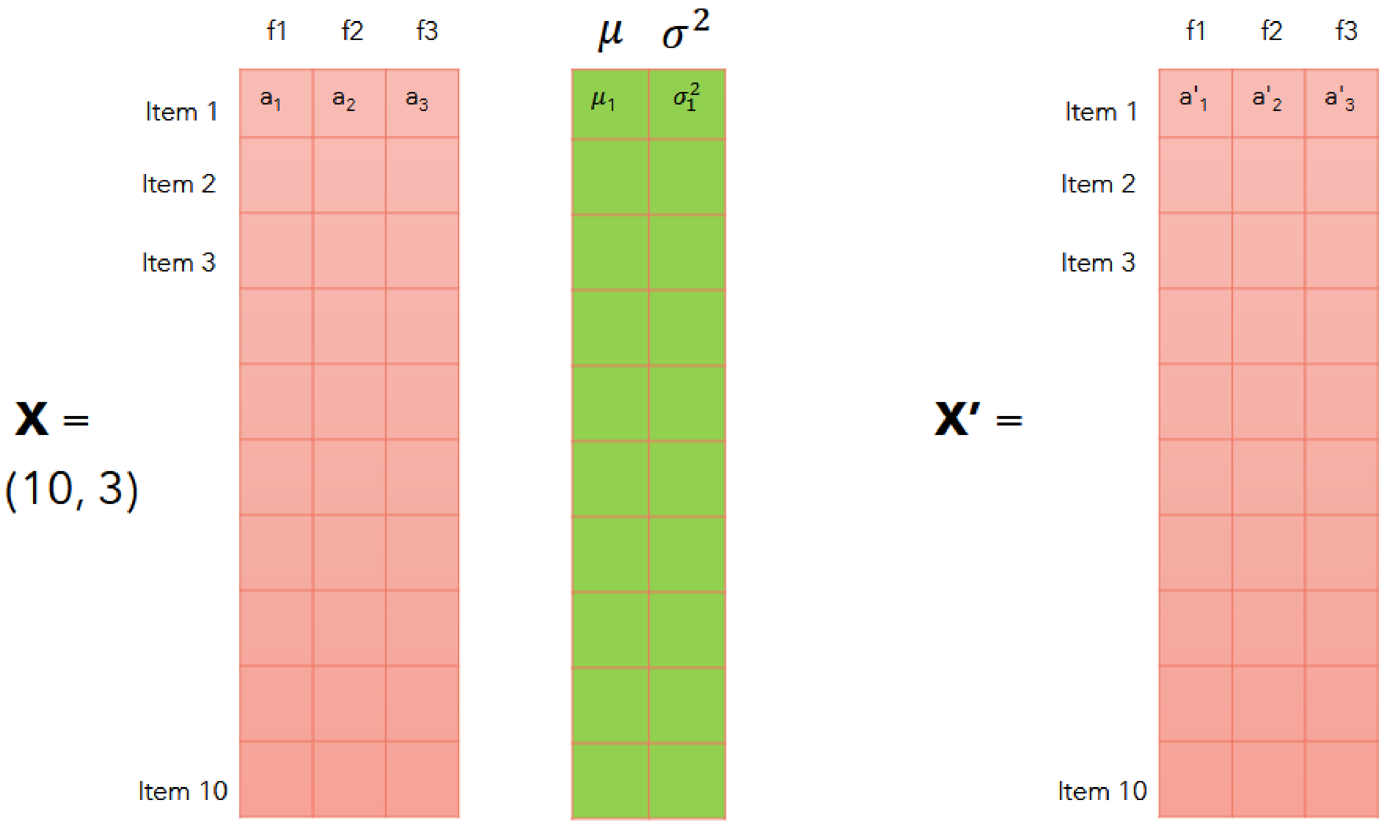
https://phontron.com/class/anlp-fall2024/assets/slides/anlp-04-attention-transformers.pdf

Recommended reading

https://jalammar.github.io/illustrated-transformer/

Layer Normalization

- **Internal Covariate Shift** occurs when the activation distributions of neurons change across training steps, forcing frequent weight adjustments.
- This phenomenon slows down training by causing instability in optimization.
- It happens when earlier layers undergo significant updates, leading to drastic changes in the inputs of subsequent layers.



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

- **Batch Normalization** normalizes across **columns (features)**.
- **Layer Normalization** normalizes across **rows (data items)**.

Root mean square normalization

Universidad EAFIT | Escuela de Ciencias Aplicadas e Ingeniería

Root Mean Square Layer Normalization

Biao Zhang[1] **Rico Sennrich**[2,1]
[1]School of Informatics, University of Edinburgh
[2]Institute of Computational Linguistics, University of Zurich
B.Zhang@ed.ac.uk, sennrich@cl.uzh.ch

4 RMSNorm

A well-known explanation of the success of LayerNorm is its re-centering and re-scaling invariance property. The former enables the model to be insensitive to shift noises on both inputs and weights, and the latter keeps the output representations intact when both inputs and weights are randomly scaled. In this paper, we hypothesize that the re-scaling invariance is the reason for success of LayerNorm, rather than re-centering invariance.

We propose RMSNorm which only focuses on re-scaling invariance and regularizes the summed inputs simply according to the root mean square (RMS) statistic:

$$\bar{a}_i = \frac{a_i}{\mathrm{RMS}(\mathbf{a})} g_i, \quad \text{where } \mathrm{RMS}(\mathbf{a}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} a_i^2}. \tag{4}$$

Intuitively, RMSNorm simplifies LayerNorm by totally removing the mean statistic in Eq. (3) at the cost of sacrificing the invariance that mean normalization affords. When the mean of summed inputs is zero, RMSNorm is exactly equal to LayerNorm. Although RMSNorm does not re-center

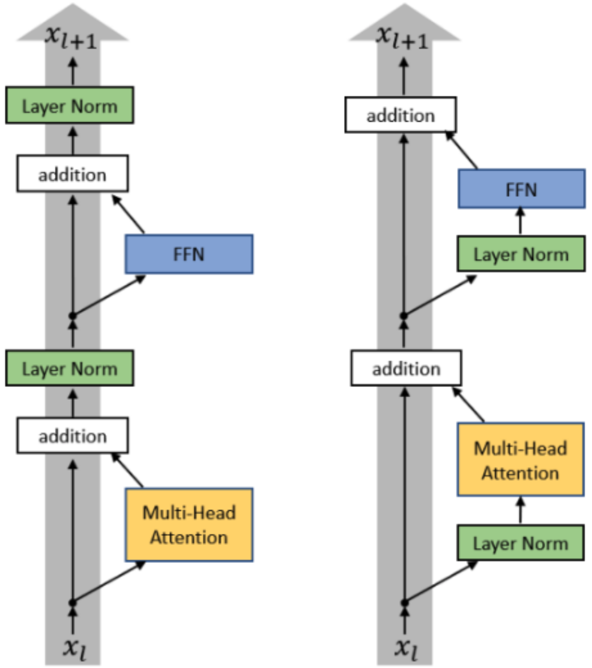
Residual Connections

- Add an additive connection between the input and output

$$\text{Residual}(\mathbf{x}, f) = f(\mathbf{x}) + \mathbf{x}$$

- Prevents vanishing gradients and allows f to learn the difference from the input
- Pre-layer-norm is better for gradient propagation

- Post-layer normalization

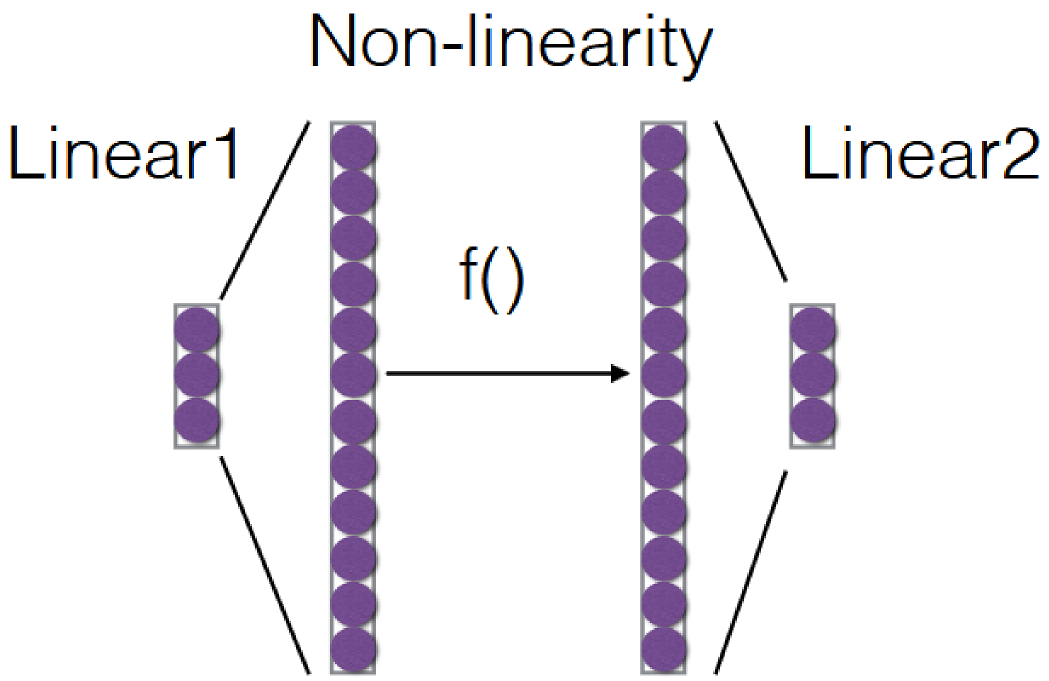


- Pre-layer normalization

Feed Forward Layers

- Extract combination features from the attended outputs

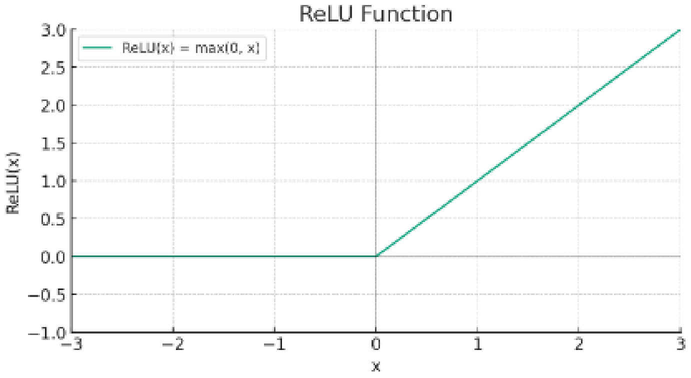
$$\mathrm{FFN}(x; W_1, \mathbf{b}_1, W_2, \mathbf{b}_2) = f(\mathbf{x}W_1 + \mathbf{b}_1)W_2 + \mathbf{b}_2$$



Feed Forward Layers

- Vaswani et al.: ReLU

$$\mathrm{ReLU}(\mathbf{x}) = \max(0, \mathbf{x})$$



- LLaMa: Swish/SiLU (Hendricks and Gimpel 2016)

$$\mathrm{Swish}(\mathbf{x}; \beta) = \mathbf{x} \odot \sigma(\beta \mathbf{x})$$