Deep Learning for Computer Vision

Self-Attention and Transformers

Vineeth N Balasubramanian

Department of Computer Science and Engineering
Indian Institute of Technology, Hyderabad

भारतीय प्रौद्योगिकी संस्थान हैदराबाद
Indian Institute of Technology Hyderabad

Review: Question

Other ways to evaluate Visual Dialog systems?

Review: Question

Other ways to evaluate Visual Dialog systems?

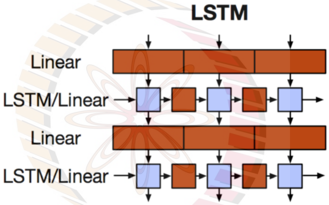
Look to NLP for consensus metrics that measure consensus between answers generated by model and a set of relevant answers; see Masciceti et al, A Revised Generative Evaluation of Visual Dialogue, arXiv 2020

Acknowledgements

- Most of this lecture's slides are based on Jay Alammar's article on "The Illustrated Transformer"
- Unless explicitly specified, assume that content and figures are either directly taken or adapted from above source

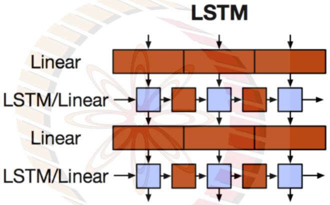
Motivation for Transformers

- Sequential computation prevents parallelization



Motivation for Transformers

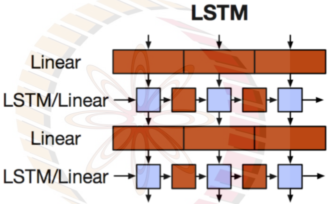
- Sequential computation prevents parallelization



- Despite GRUs and LSTMs, RNNs still need attention mechanism to deal with long-range dependencies – path length for co-dependent computation between states grows with sequence length

Motivation for Transformers

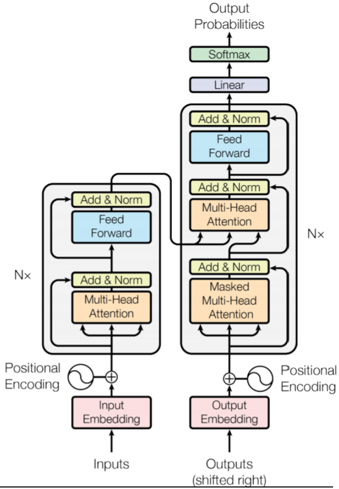
- Sequential computation prevents parallelization



- Despite GRUs and LSTMs, RNNs still need attention mechanism to deal with long-range dependencies – path length for co-dependent computation between states grows with sequence length
- But if attention gives us access to any state, maybe we don't need the RNN?!

Credits: Richard Socher (Stanford CS224n)

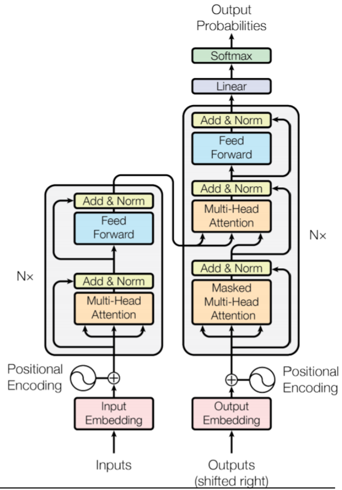
Transformers[1]



- The work "Attention is All you Need" (Vaswani et al, NeurIPS 2017) first made it possible to do Seq2Seq modeling without RNNs

[1] Vaswani et al, Attention is All You Need, NeurIPS 2017

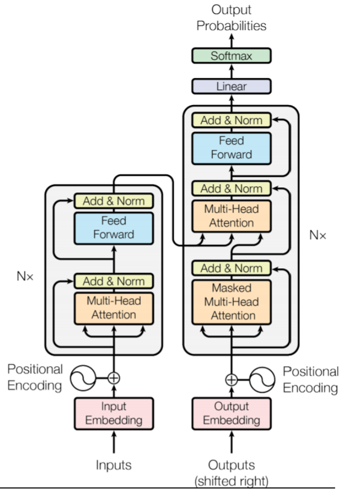
Transformers[1]



- The work "Attention is All you Need" (Vaswani et al, NeurIPS 2017) first made it possible to do Seq2Seq modeling without RNNs
- Proposed **transformer model**, entirely built on **self-attention mechanism** without using sequence-aligned recurrent architectures

[1] Vaswani et al, Attention is All You Need, NeurIPS 2017

Transformers[1]

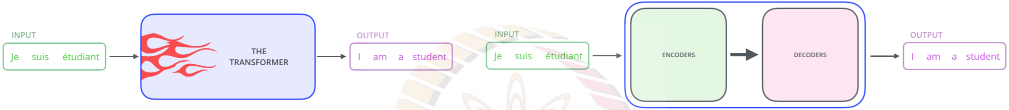


- The work "Attention is All you Need" (Vaswani et al, NeurIPS 2017) first made it possible to do Seq2Seq modeling without RNNs
- Proposed **transformer model**, entirely built on **self-attention mechanism** without using sequence-aligned recurrent architectures
- Key components:
  - Self-Attention
  - Multi-Head Attention
  - Positional Encoding
  - Encoder-Decoder Architecture

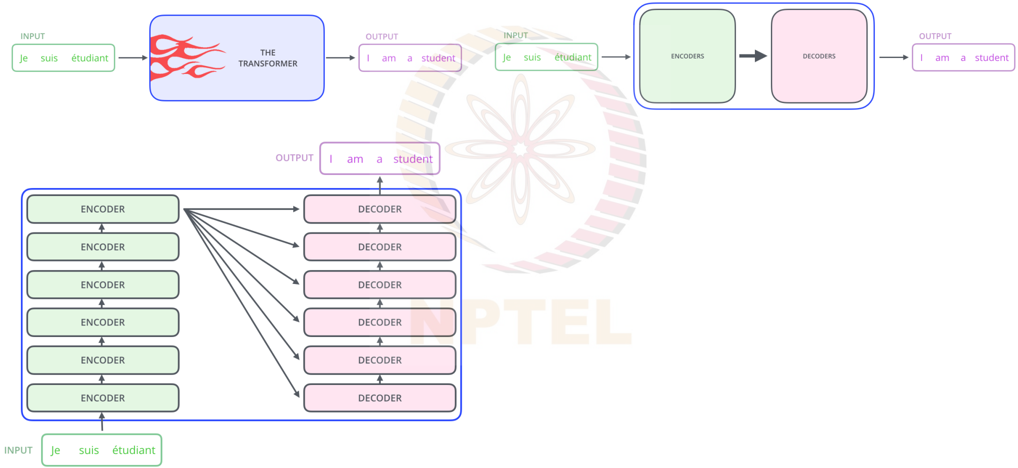
[1]Vaswani et al, Attention is All You Need, NeurIPS 2017

Transformers in a Nutshell

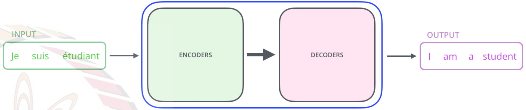
Transformers in a Nutshell

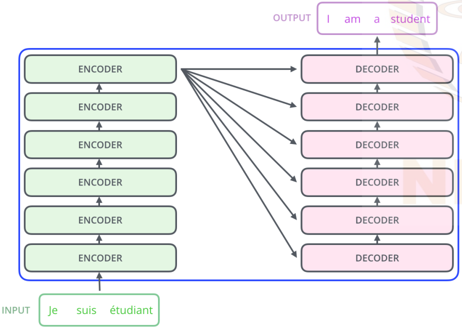


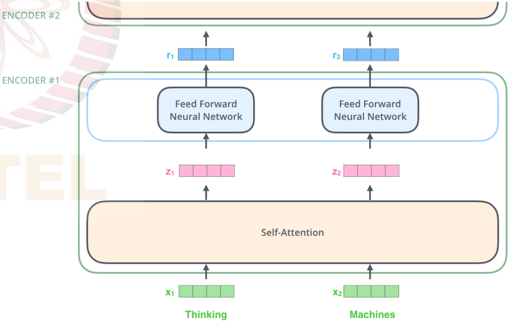
Transformers in a Nutshell



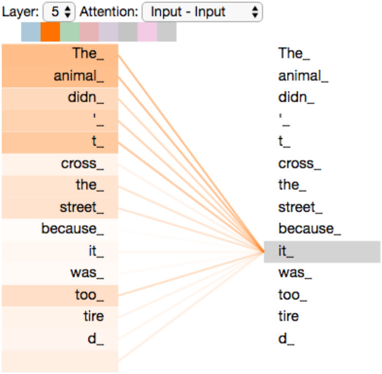
Transformers in a Nutshell





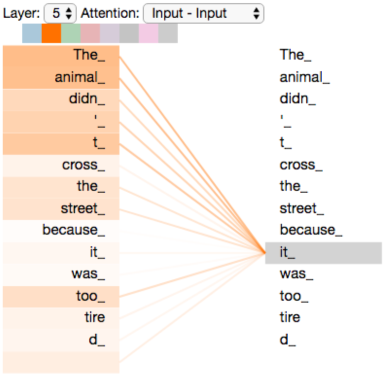


Self-Attention



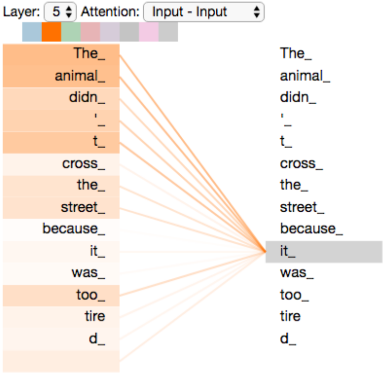
- Consider two input sentences we want to translate:

Self-Attention



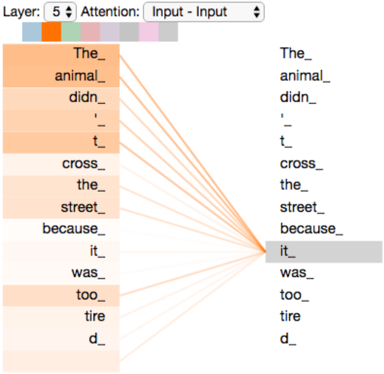
- Consider two input sentences we want to translate:
  - *The animal didn't cross the street because it was too tired*

Self-Attention



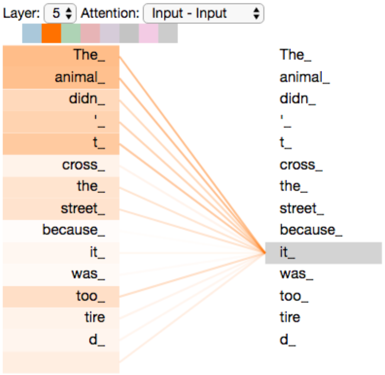
- Consider two input sentences we want to translate:
  - *The animal didn't cross the street because it was too tired*
  - *The animal didn't cross the street because it was too wide*

Self-Attention



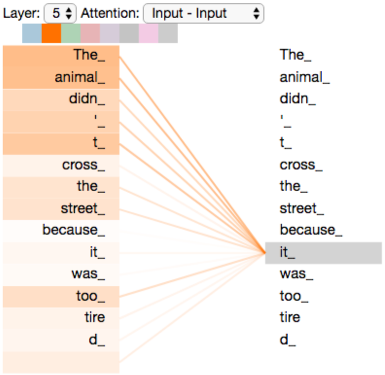
- Consider two input sentences we want to translate:
  - *The animal didn't cross the street because it was too tired*
  - *The animal didn't cross the street because it was too wide*
- "it" refers to "animal" in first case, but to "street" in second case; this is hard for traditional Seq2Seq models to model

Self-Attention



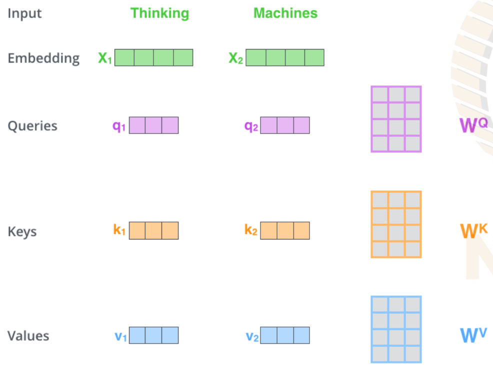
- Consider two input sentences we want to translate:
  - *The animal didn't cross the street because it was too tired*
  - *The animal didn't cross the street because it was too wide*
- "it" refers to "animal" in first case, but to "street" in second case; this is hard for traditional Seq2Seq models to model
- As the model processes each word, self-attention allows it to look at other positions in input sequence to help get a better encoding

Self-Attention



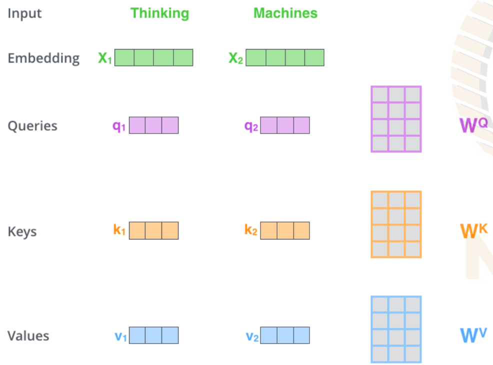
- Consider two input sentences we want to translate:
  - *The animal didn't cross the street because it was too tired*
  - *The animal didn't cross the street because it was too wide*
- "it" refers to "animal" in first case, but to "street" in second case; this is hard for traditional Seq2Seq models to model
- As the model processes each word, self-attention allows it to look at other positions in input sequence to help get a better encoding
- Recall RNNs: we now no longer need to maintain a hidden state to incorporate representation of previous words/vectors!

Self-Attention



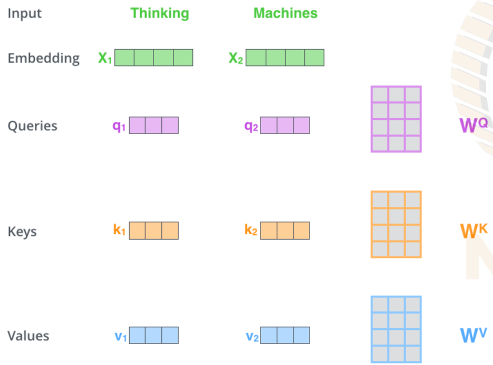
- **STEP 1:** Create three vectors from encoder's input vector (x_i):
  - Query vector (q_i)
  - Key vector (k_i)
  - Value vector (v_i)

Self-Attention



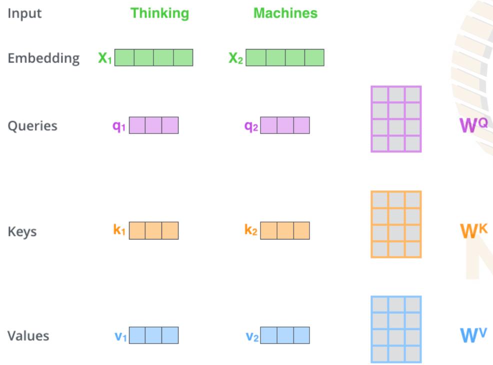
- **STEP 1:** Create three vectors from encoder's input vector (x_i):
  - Query vector (q_i)
  - Key vector (k_i)
  - Value vector (v_i)
- These are created by multiplying input with weight matrices W^Q, W^K, W^V, learned during training

Self-Attention



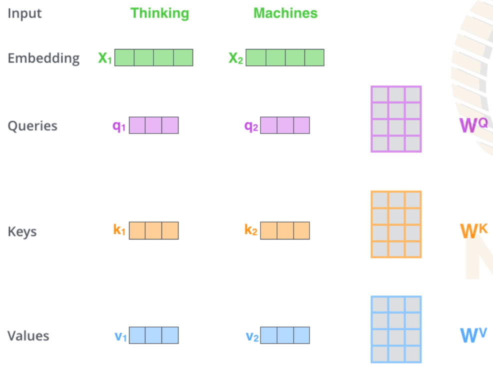
- **STEP 1:** Create three vectors from encoder's input vector (x_i):
  - Query vector (q_i)
  - Key vector (k_i)
  - Value vector (v_i)
- These are created by multiplying input with weight matrices W^Q, W^K, W^V, learned during training
- In the paper, $q, k, v \in \mathbb{R}^{64}$ and $x \in \mathbb{R}^{512}$

Self-Attention



- **STEP 1:** Create three vectors from encoder's input vector (x_i):
  - Query vector (q_i)
  - Key vector (k_i)
  - Value vector (v_i)
- These are created by multiplying input with weight matrices W^Q, W^K, W^V, learned during training
- In the paper, $q, k, v \in \mathbb{R}^{64}$ and $x \in \mathbb{R}^{512}$
- Do q, k, v always have to be smaller than x?

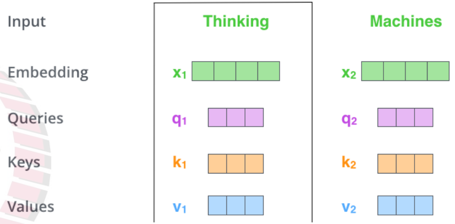
Self-Attention



- **STEP 1:** Create three vectors from encoder's input vector (x_i):
  - Query vector (q_i)
  - Key vector (k_i)
  - Value vector (v_i)
- These are created by multiplying input with weight matrices W^Q, W^K, W^V, learned during training
- In the paper, $q, k, v \in \mathbb{R}^{64}$ and $x \in \mathbb{R}^{512}$
- Do q, k, v always have to be smaller than x? No, this was done perhaps to make computation of multi-headed attention constant
- What are the dimensions of W^Q, W^K, W^V?

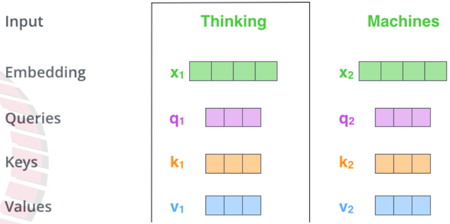
Self-Attention

- **STEP 2:** Calculate self-attention scores - score all words of input sentence against themselves; how?



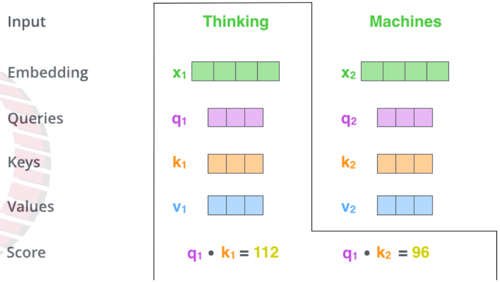
Self-Attention

- **STEP 2:** Calculate self-attention scores - score all words of input sentence against themselves; how?
- By taking dot product of **query vector** with **key vector** of respective words



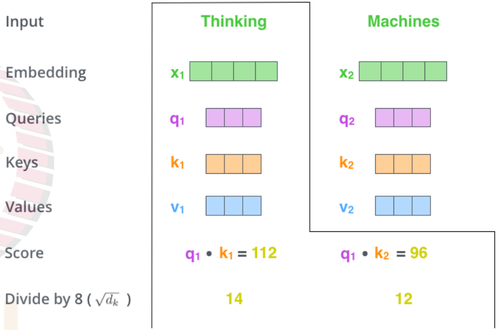
Self-Attention

- **STEP 2:** Calculate self-attention scores - score all words of input sentence against themselves; how?
- By taking dot product of **query vector** with **key vector** of respective words
- E.g. for input "Thinking", first score would be $q_1 \cdot k_1$ (with itself); second score would be dot product of $q_1 \cdot k_2$ (with "Machines"), and so on



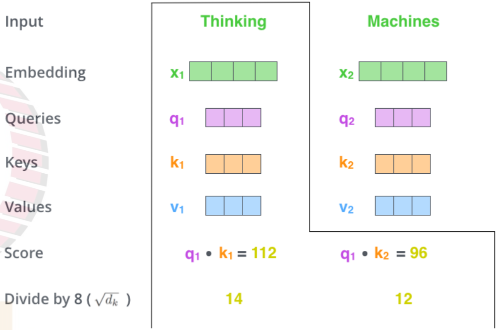
Self-Attention

- **STEP 2:** Calculate self-attention scores - score all words of input sentence against themselves; how?
- By taking dot product of **query vector** with **key vector** of respective words
- E.g. for input "Thinking", first score would be $q_1 \cdot k_1$ (with itself); second score would be dot product of $q_1 \cdot k_2$ (with "Machines"), and so on
- Scores then divided by $\sqrt{length(k)}$



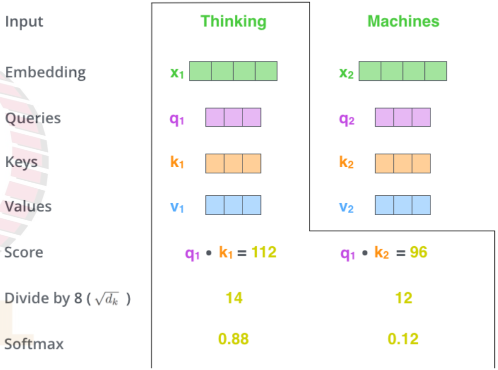
Self-Attention

- **STEP 2:** Calculate self-attention scores - score all words of input sentence against themselves; how?
- By taking dot product of **query vector** with **key vector** of respective words
- E.g. for input "Thinking", first score would be $q_1 \cdot k_1$ (with itself); second score would be dot product of $q_1 \cdot k_2$ (with "Machines"), and so on
- Scores then divided by $\sqrt{length(k)}$
- This is **Scaled Dot-Product Attention**, recall from W9P1; this design choice leads to more stable gradients



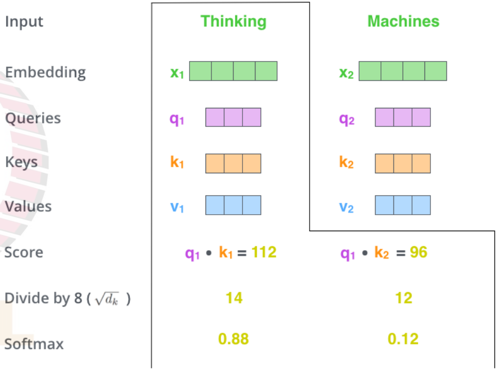
Self-Attention

- **STEP 3:** Softmax used to get normalized probability scores; determines how much each word will be expressed at this position



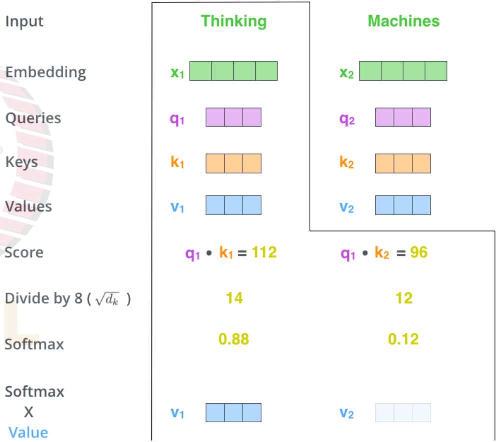
Self-Attention

- **STEP 3:** Softmax used to get normalized probability scores; determines how much each word will be expressed at this position
- Clearly, word at this position will have highest softmax score, but sometimes it's useful to attend to another word that is relevant



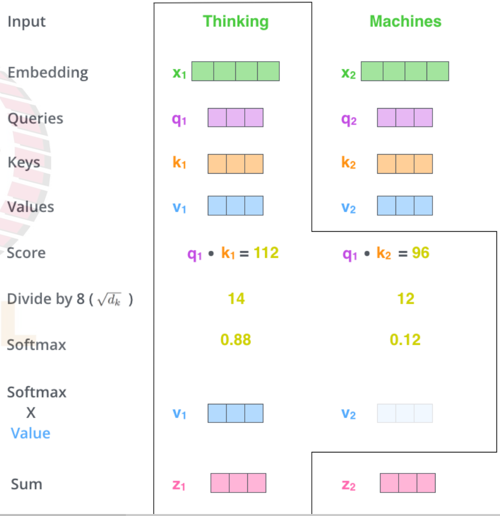
Self-Attention

- **STEP 3:** Softmax used to get normalized probability scores; determines how much each word will be expressed at this position
- Clearly, word at this position will have highest softmax score, but sometimes it's useful to attend to another word that is relevant
- **STEP 4:** Multiply each **value vector** by softmax score; why? Keep values of word(s) we want to focus on intact, and drown out irrelevant words

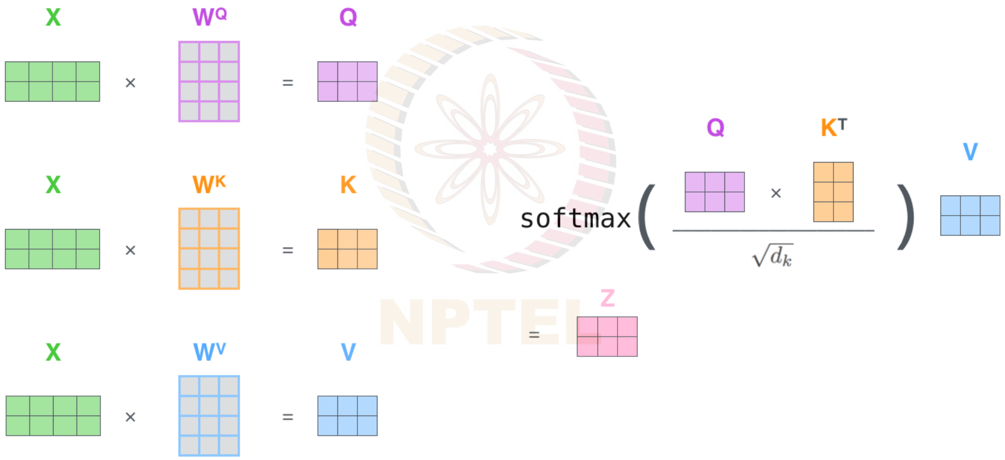


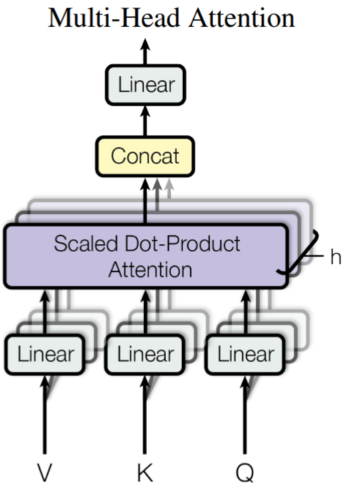
Self-Attention

- **STEP 3:** Softmax used to get normalized probability scores; determines how much each word will be expressed at this position
- Clearly, word at this position will have highest softmax score, but sometimes it's useful to attend to another word that is relevant
- **STEP 4:** Multiply each **value vector** by softmax score; why? Keep values of word(s) we want to focus on intact, and drown out irrelevant words
- **STEP 5:** Sum up weighted value vectors $\rightarrow$ produces output of self-attention layer at this position (for first word)



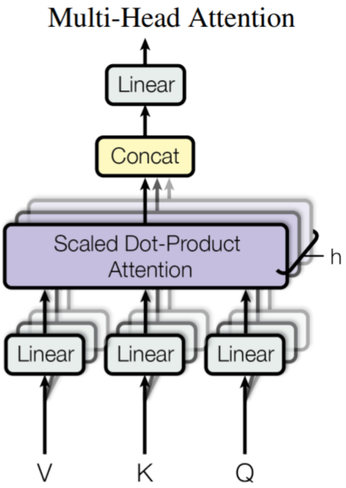
Self-Attention: Illustration





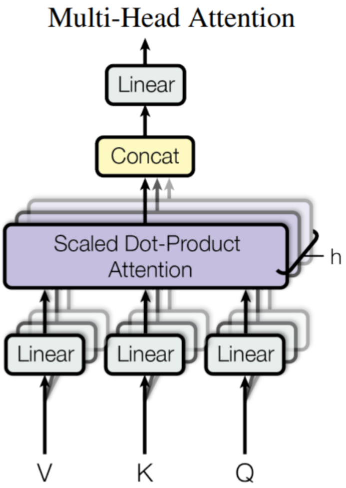
- Improves performance of the attention layer in two ways:

Multi-Head Attention



- Improves performance of the attention layer in two ways:
  - Expands model's ability to focus on different positions. In example above, z_1 contains a bit of every other encoding, but dominated by actual word itself

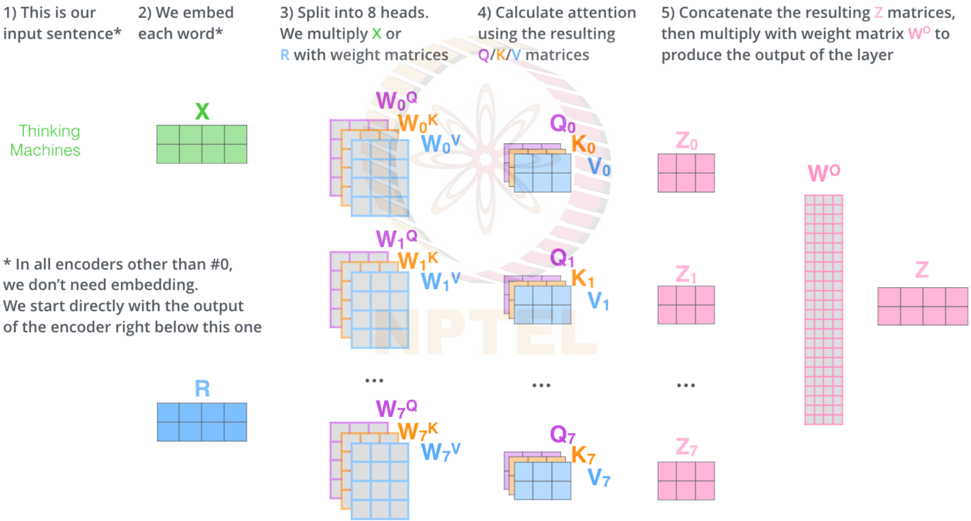
Multi-Head Attention



- Improves performance of the attention layer in two ways:
  - Expands model's ability to focus on different positions. In example above, z_1 contains a bit of every other encoding, but dominated by actual word itself
  - Gives attention layer multiple "*representation subspaces*"; we have not one, but multiple sets of Query/Key/Value weight matrices; after training, each set is used to project input embeddings into different representation subspaces

Credit: Vaswani et al, Attention is All You Need, NeurIPS 2017

Multi-Head Attention: Illustration



Positional Encoding

- Unlike RNN and CNN encoders, attention encoder outputs do not depend on order of inputs (Why?)

Positional Encoding

- Unlike RNN and CNN encoders, attention encoder outputs do not depend on order of inputs (Why?)
- But order of sequence conveys important information for machine translation tasks and language modeling

Positional Encoding

- Unlike RNN and CNN encoders, attention encoder outputs do not depend on order of inputs (Why?)
- But order of sequence conveys important information for machine translation tasks and language modeling
- The idea: Add positional information of input token in the sequence into input embedding vectors

$$PE_{pos,2i} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{emb}}}}\right) \qquad PE_{pos,2i+1} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{emb}}}}\right)$$

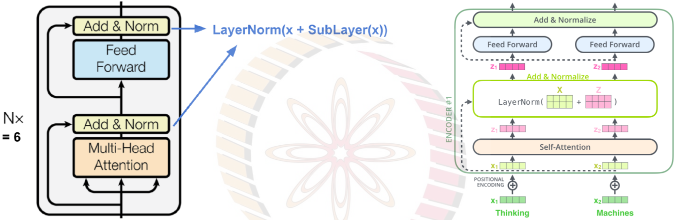
Positional Encoding

- Unlike RNN and CNN encoders, attention encoder outputs do not depend on order of inputs (Why?)
- But order of sequence conveys important information for machine translation tasks and language modeling
- The idea: Add positional information of input token in the sequence into input embedding vectors

$$PE_{pos,2i} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{emb}}}}\right) \qquad PE_{pos,2i+1} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{emb}}}}\right)$$

- Final input embeddings are concatenation of learnable embedding and positional encoding

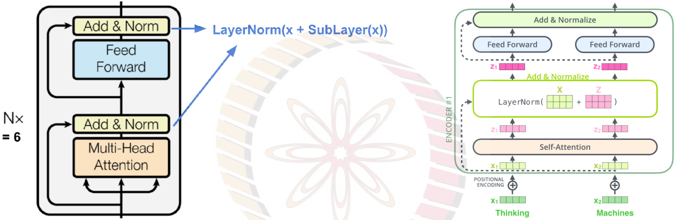
Encoder



- Stack of N=6 identical layers

Credit: "Attention? Attention!" by Lilian Weng

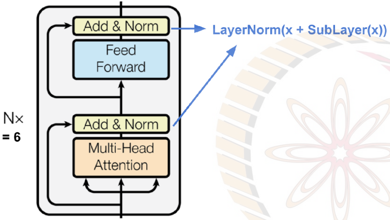
Encoder

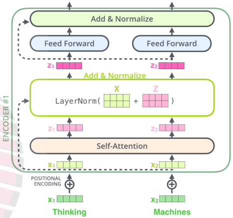


- Stack of N=6 identical layers
- Each layer has a **multi-head self-attention layer** and a simple position-wise fully connected **feedforward network**

Credit: "Attention? Attention!" by Lilian Weng

Encoder

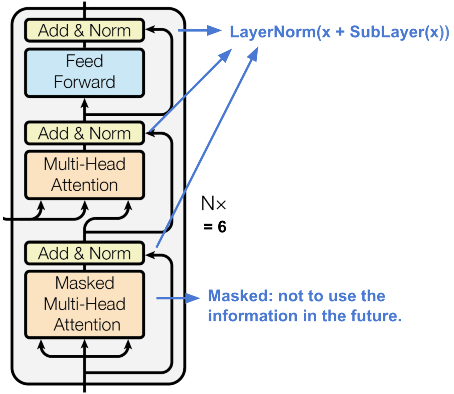




- Stack of N=6 identical layers
- Each layer has a **multi-head self-attention layer** and a simple position-wise fully connected **feedforward network**
- Each sub-layer has a **residual** connection and **layer-normalization**; all sub-layers output data of same dimension $d_{model} = 512$

Credit: "Attention? Attention!" by Lilian Weng

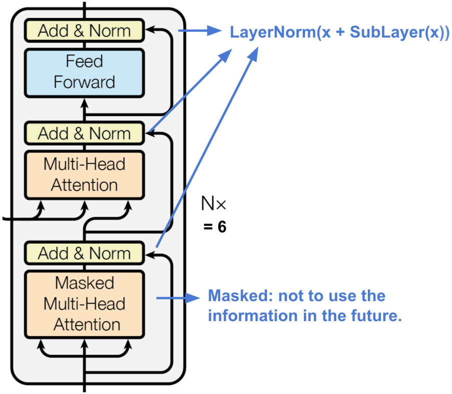
Decoder



- Stack of **N=6** identical layers

Credit: "Attention? Attention!" by Lilian Weng

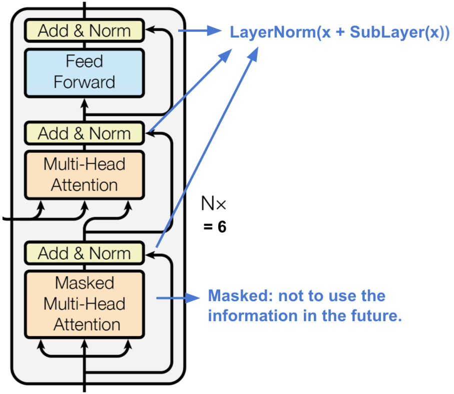
Decoder



- Stack of **N=6** identical layers
- Each layer has two sub-layers of **multi-head attention** mechanisms and one sub-layer of fully-connected **feedforward network**

Credit: "Attention? Attention!" by Lilian Weng

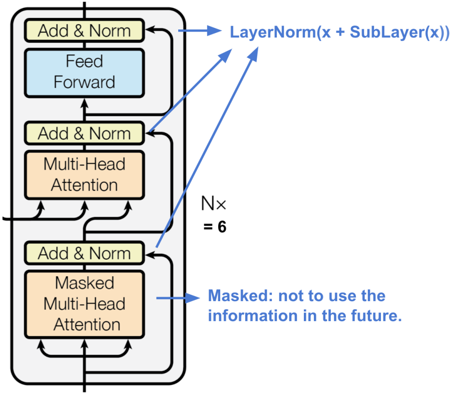
Decoder



- Stack of **N=6** identical layers
- Each layer has two sub-layers of **multi-head attention** mechanisms and one sub-layer of fully-connected **feedforward network**
- Similar to encoder, each sub-layer adopts a **residual connection** and a **layer-normalization**

Credit: "Attention? Attention!" by Lilian Weng

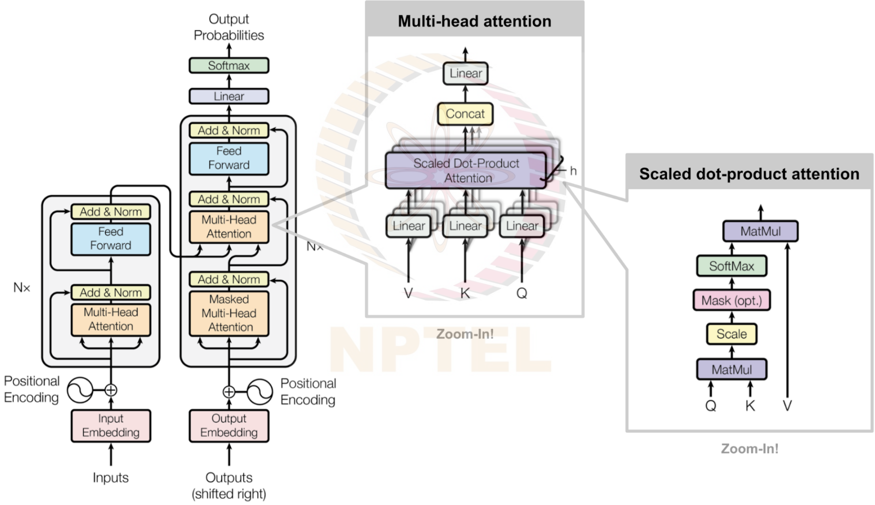
Decoder



- Stack of **N=6** identical layers
- Each layer has two sub-layers of **multi-head attention** mechanisms and one sub-layer of fully-connected **feedforward network**
- Similar to encoder, each sub-layer adopts a **residual connection** and a **layer-normalization**
- First multi-head attention sub-layer is modified to prevent positions from attending to subsequent positions, as we don't want to look into future of target sequence when predicting current position

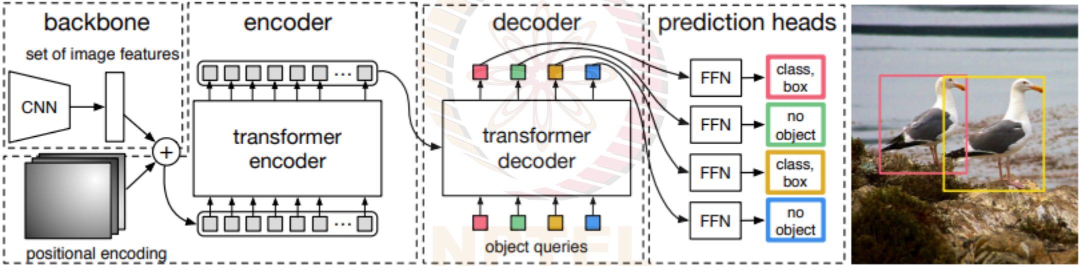
Credit: "Attention? Attention!" by Lilian Weng

Transformers: Full Architecture



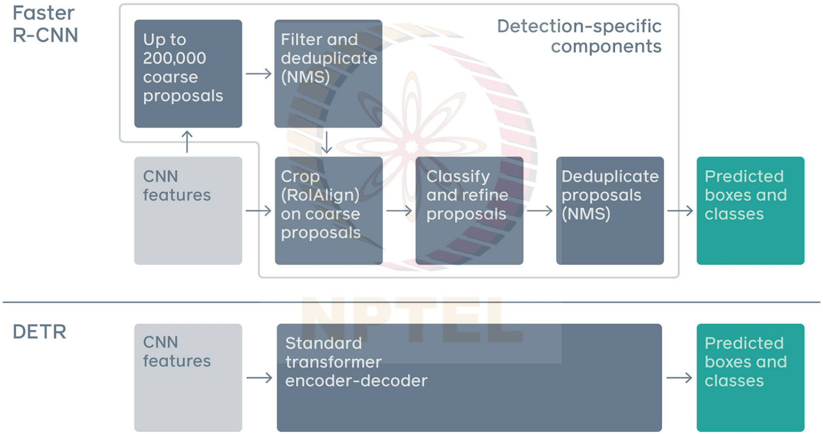
Credit: "Attention? Attention!" by Lilian Weng

Transformers in Computer Vision: Object Detection[2]



[2]Carion et al, End-to-End Object Detection with Transformers, ECCV 2020

Transformers in Computer Vision: Object Detection



Credit: *Ram Sagar, Analytics India Mag*

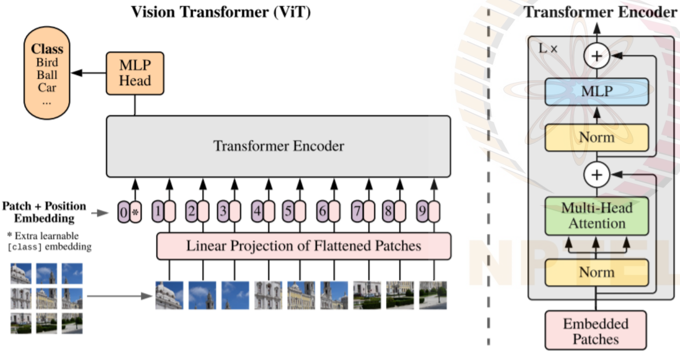
Transformers in Computer Vision: Object Detection[3]

Results on MS COCO validation set

Model	GFLOPS/FPS	#params	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Faster RCNN-DC5	320/16	166M	39.0	60.5	42.3	21.4	43.5	52.5
Faster RCNN-FPN	180/26	42M	40.2	61.0	43.8	24.2	43.5	52.0
Faster RCNN-R101-FPN	246/20	60M	42.0	62.5	45.9	25.2	45.6	54.6
Faster RCNN-DC5+	320/16	166M	41.1	61.4	44.3	22.9	45.9	55.0
Faster RCNN-FPN+	180/26	42M	42.0	62.1	45.5	26.6	45.4	53.4
Faster RCNN-R101-FPN+	246/20	60M	44.0	63.9	**47.8**	**27.2**	48.1	56.0
DETR	86/28	41M	42.0	62.4	44.2	20.5	45.8	61.1
DETR-DC5	187/12	41M	43.3	63.1	45.9	22.5	47.3	61.1
DETR-R101	152/20	60M	43.5	63.8	46.4	21.9	48.0	61.8
DETR-DC5-R101	253/10	60M	**44.9**	**64.7**	47.7	23.7	**49.5**	**62.3**

[3] Carion et al, End-to-End Object Detection with Transformers, ECCV 2020

Transformers in Computer Vision: Image Recognition[4]



Credit: *Nabil Madali, Gitconnected*

- Image split into fixed-size patches
- Each of them linearly embedded
- Position embeddings added to resulting sequence of vectors
- Patches fed to standard Transformer encoder
- In order to perform classification, standard approach of adding an extra learnable "classification token" added to sequence

[4]Dosovitskiy et al, An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale, arXiv 2020

Homework

Readings

- Watch the Transformers in Action video provided in the week's lecture materials
- The Illustrated Transformer article by Jay Alammar
- A detailed explanation of positional encoding by Amirhossein Kazemnejad
- For more information: Attention is All You Need paper by Vaswani, et al. (NeurIPS 2017)

Questions

- Are transformers faster or slower than LSTMs? What is the reason for your opinion?