Attention and Transformers

Vineeth N Balasubramanian

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

6 Dec 2022

Attention and Transformers

What is this session about?

- One of the hottest topics in DL today
- An introduction to attention in DL
- A deeper understanding of the original 'Transformer' architecture
- A few variants and newer applications
- (Why are transformers a good stand-up comedy material?)

Attention and Transformers

What is this session about?

- One of the hottest topics in DL today
- An introduction to attention in DL
- A deeper understanding of the original 'Transformer' architecture
- A few variants and newer applications
- (Why are transformers a good stand-up comedy material?)

What is this session NOT about?

- We will not go too much into recent transformer variants (there are just too many!)
- May not have bandwidth to cover code implementation
- Disclaimer: My knowledge of NLP is sub-par!

Attention in the Pre-Transformer Era

- In psychology, attention is defined as the cognitive ability of humans to focus on few relevant things while processing a lot of information

Attention in the Pre-Transformer Era

- In psychology, attention is defined as the cognitive ability of humans to focus on few relevant things while processing a lot of information
- Machine learning models, especially neural networks, are designed to mimic the working of human brain

Attention in the Pre-Transformer Era

- In psychology, attention is defined as the cognitive ability of humans to focus on few relevant things while processing a lot of information
- Machine learning models, especially neural networks, are designed to mimic the working of human brain
- Attention mechanism in neural networks attempts to execute the same task of focusing on few important things from among many.

Attention in the Pre-Transformer Era

- In psychology, attention is defined as the cognitive ability of humans to focus on few relevant things while processing a lot of information
- Machine learning models, especially neural networks, are designed to mimic the working of human brain
- Attention mechanism in neural networks attempts to execute the same task of focusing on few important things from among many.

Visual Selective Attention[1]

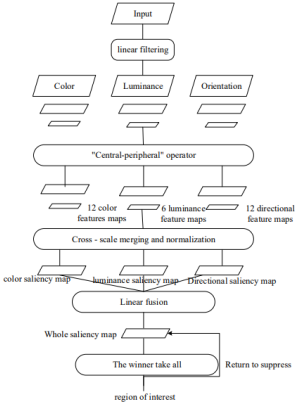


Figure 1: Visual attention computation model proposed by Itti and Koch

- Attention modeled as a way to focus on parts of images

[1] Itti, Koch, Niebur. A model of saliency-based visual attention for rapid scene analysis. IEEE TPAMI 2002

Visual Selective Attention[1]

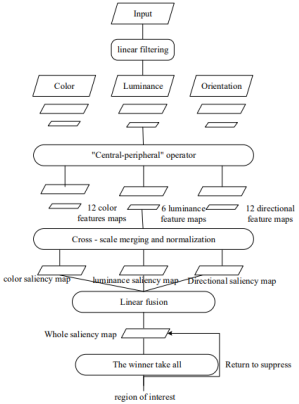


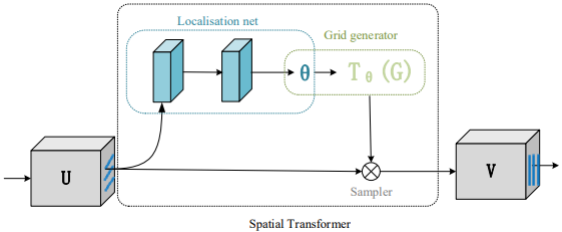
Figure 1: Visual attention computation model proposed by Itti and Koch

- Attention modeled as a way to focus on parts of images
- Notions of *top-down* and *bottom-up* attention

[1] Itti, Koch, Niebur. A model of saliency-based visual attention for rapid scene analysis. IEEE TPAMI 2002

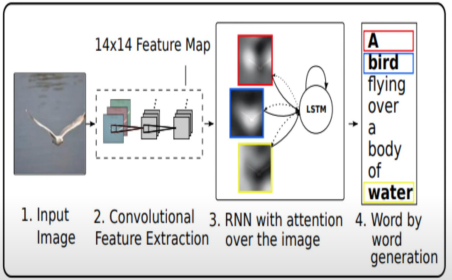
More Recent Notions of Attention[2]

Spatial Transformer Networks



[2] Jaderberg et al, Spatial Transformer Networks, NeurIPS 2015

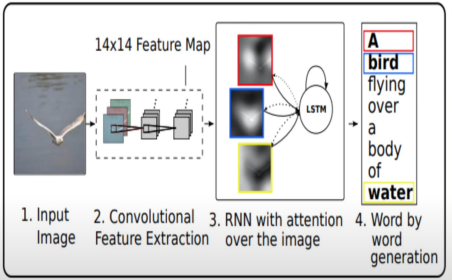
Attention in Image Captioning[3]



- Generate captions from an input image using the attention mechanism.

[3]Xu et al, Show, Attend and Tell: Neural Image Caption Generation with Visual Attention, ICML 2015

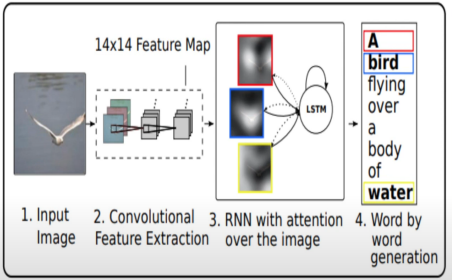
Attention in Image Captioning[3]



- Generate captions from an input image using the attention mechanism.
- VGGnet as encoder for extracting features from input image

[3]Xu et al, Show, Attend and Tell: Neural Image Caption Generation with Visual Attention, ICML 2015

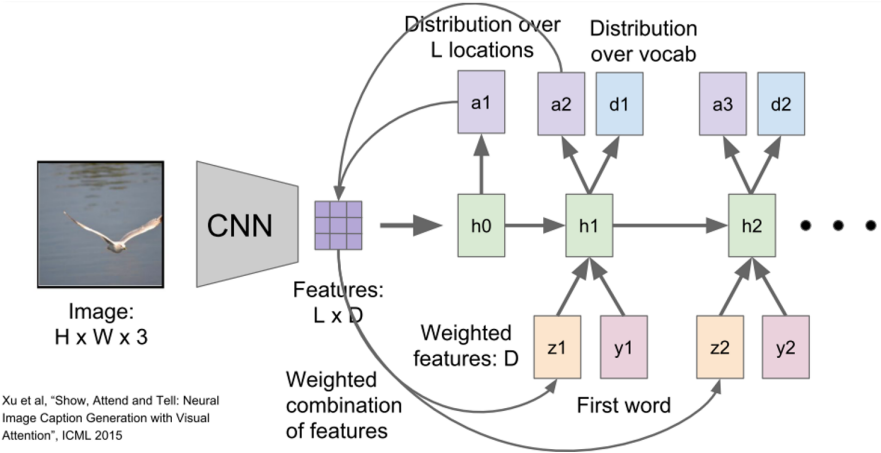
Attention in Image Captioning[3]



- Generate captions from an input image using the attention mechanism.
- VGGnet as encoder for extracting features from input image
- LSTM decoder generates captions by producing one word at each time step by using the context vector, previous hidden state and previously generated words.

[3]Xu et al, Show, Attend and Tell: Neural Image Caption Generation with Visual Attention, ICML 2015

Attention in Image Captioning



Context Vector

- The context vector $\hat{\mathbf{z}}_t$ is computed by a function ϕ using the attention vectors $\mathbf{a}_i$ and the positive weights α_i, $i = 1, 2, ..., L$ correspond to different locations of the input image.

$$e_{ti} = f_{att}(\mathbf{a}_i, \mathbf{h}_{t-1})$$
$$\alpha_{ti} = \frac{exp(e_{ti})}{\sum_{k=1}^{L} exp(e_{tk})}$$
$$\hat{\mathbf{z}}_t = \phi(\{\mathbf{a}_i\}, \{\alpha_i\})$$

Context Vector

- The context vector $\hat{\mathbf{z}}_t$ is computed by a function ϕ using the attention vectors $\mathbf{a}_i$ and the positive weights α_i, $i = 1, 2, ..., L$ correspond to different locations of the input image.

$$e_{ti} = f_{att}(\mathbf{a}_i, \mathbf{h}_{t-1})$$
$$\alpha_{ti} = \frac{exp(e_{ti})}{\sum_{k=1}^{L} exp(e_{tk})}$$
$$\hat{\mathbf{z}}_t = \phi(\{\mathbf{a}_i\}, \{\alpha_i\})$$

- At each location i, the weight α_i for the annotation vector $\mathbf{a}_i$ is calculated by an attention mechanism f_{att}.

Context Vector

- The context vector $\hat{\mathbf{z}}_t$ is computed by a function ϕ using the attention vectors $\mathbf{a}_i$ and the positive weights α_i, $i = 1, 2, ..., L$ correspond to different locations of the input image.

$$e_{ti} = f_{att}(\mathbf{a}_i, \mathbf{h}_{t-1})$$
$$\alpha_{ti} = \frac{exp(e_{ti})}{\sum_{k=1}^{L} exp(e_{tk})}$$
$$\hat{\mathbf{z}}_t = \phi(\{\mathbf{a}_i\}, \{\alpha_i\})$$

- At each location i, the weight α_i for the annotation vector $\mathbf{a}_i$ is calculated by an attention mechanism f_{att}.
- α_i can be interpreted in two ways based on the two versions of this attention mechanism, namely the stochastic "**Hard**" and the deterministic "**Soft**".

Credits: Xu et al, Show, Attend and Tell: Neural Image Caption Generation with Visual Attention, ICML 2015

Hard vs Soft Attention

- In **hard attention**, to generate the word $\mathbf{y}_i$, the weight α_i is seen as the probability that location i is the correct place to focus on.
- s_t is defined as the location variable that determines where the attention should be focused while producing the t^{th} word.
- They assign a multinoulli distribution with weights α_i as the parameters and view $\hat{\mathbf{z}}_t$ as a random variable

$$p(s_{ti} = 1 | s_{j<t}, \mathbf{a}) = \alpha_{t,i}$$

$$\hat{\mathbf{z}}_t = \sum_t s_{t,i} \mathbf{a}_i$$

- We can't simply train the model with back propagation while using hard attention as we probabilistically sample from one of the attention vectors $\mathbf{a}_i$ and use that as the context vector $\hat{\mathbf{z}}_t$.

Credits: Xu et al, Show, Attend and Tell: Neural Image Caption Generation with Visual Attention, ICML 2015

Hard vs Soft Attention

- In **hard attention**, to generate the word $\mathbf{y}_i$, the weight α_i is seen as the probability that location i is the correct place to focus on.
- s_t is defined as the location variable that determines where the attention should be focused while producing the t^{th} word.
- They assign a multinoulli distribution with weights α_i as the parameters and view $\hat{\mathbf{z}}_t$ as a random variable

$$p(s_{ti} = 1 | s_{j<t}, \mathbf{a}) = \alpha_{t,i}$$
$$\hat{\mathbf{z}}_t = \sum_t s_{t,i} \mathbf{a}_i$$

- We can't simply train the model with back propagation while using hard attention as we probabilistically sample from one of the attention vectors $\mathbf{a}_i$ and use that as the context vector $\hat{\mathbf{z}}_t$.

Credits: Xu et al, Show, Attend and Tell: Neural Image Caption Generation with Visual Attention, ICML 2015

Hard vs Soft Attention

- In **hard attention**, to generate the word $\mathbf{y}_i$, the weight α_i is seen as the probability that location i is the correct place to focus on.
- s_t is defined as the location variable that determines where the attention should be focused while producing the t^{th} word.
- They assign a multinoulli distribution with weights α_i as the parameters and view $\hat{\mathbf{z}}_t$ as a random variable

$$p(s_{ti} = 1 | s_{j<t}, \mathbf{a}) = \alpha_{t,i}$$
$$\hat{\mathbf{z}}_t = \sum_t s_{t,i} \mathbf{a}_i$$

- We can't simply train the model with back propagation while using hard attention as we probabilistically sample from one of the attention vectors $\mathbf{a}_i$ and use that as the context vector $\hat{\mathbf{z}}_t$.

Credits: Xu et al, Show, Attend and Tell: Neural Image Caption Generation with Visual Attention, ICML 2015

Hard vs Soft Attention

- In **soft attention**, to generate the word $\mathbf{y}_i$, the weight α_i is seen as the relative importance given to location i in combining the attention vectors $\mathbf{a}_i's$.
- Instead of sampling the location vector s_t every time as in the hard attention, an expectation of the context vector $\hat{\mathbf{z}}_t$ is taken and a deterministic attention model is formulated as:

$$\mathbb{E}_{p(s_t|a)}[\hat{\mathbf{z}}_t] = \sum_{i=1}^{L} \alpha_{t,i}\mathbf{a}_i$$

$$\phi(\{\mathbf{a}_i\}, \{\alpha_i\}) = \sum_{i}^{L} \alpha_i \mathbf{a}_i$$

- As this model is differentiable under the deterministic attention, it can be trained end-to-end using back propagation.

Credits: Xu et al, Show, Attend and Tell: Neural Image Caption Generation with Visual Attention, ICML 2015

Hard vs Soft Attention

- In **soft attention**, to generate the word $\mathbf{y}_i$, the weight α_i is seen as the relative importance given to location i in combining the attention vectors $\mathbf{a}_i's$.
- Instead of sampling the location vector s_t every time as in the hard attention, an expectation of the context vector $\hat{\mathbf{z}}_t$ is taken and a deterministic attention model is formulated as:

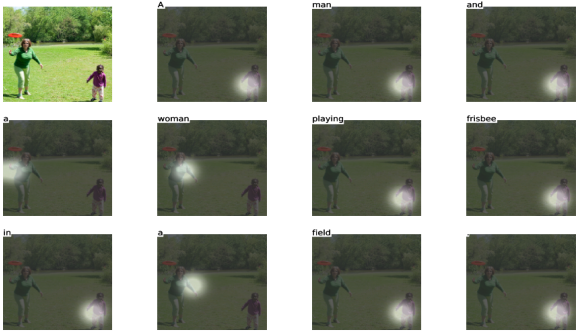
$$\mathbb{E}_{p(s_t|a)}[\hat{\mathbf{z}}_t] = \sum_{i=1}^{L} \alpha_{t,i}\mathbf{a}_i$$

$$\phi(\{\mathbf{a}_i\}, \{\alpha_i\}) = \sum_{i}^{L} \alpha_i \mathbf{a}_i$$

- As this model is differentiable under the deterministic attention, it can be trained end-to-end using back propagation.

Credits: Xu et al, Show, Attend and Tell: Neural Image Caption Generation with Visual Attention, ICML 2015

Hard vs Soft Attention

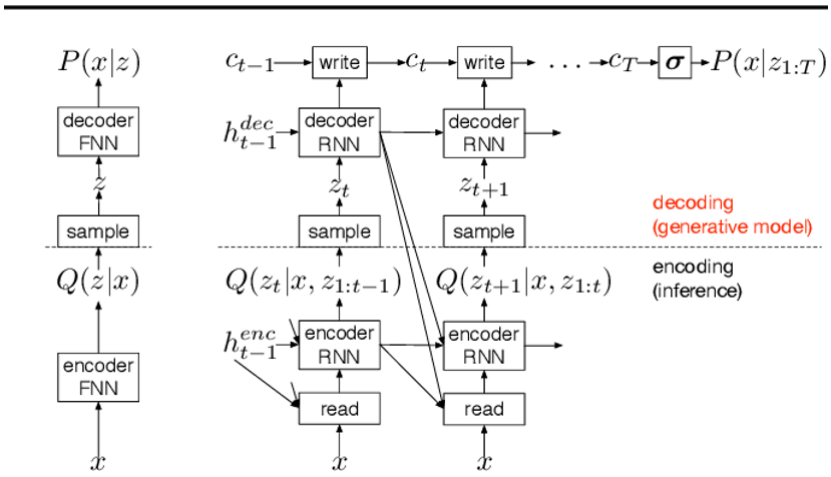


(a) A man and a woman playing frisbee in a field.

(b) A woman is throwing a frisbee in a park.

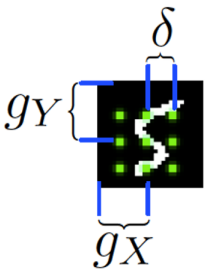
Credits: Xu et al, Show, Attend and Tell: Neural Image Caption Generation with Visual Attention, PMLR 2015.

DRAW: Deep Recurrent Attentive Writer[4]



[4]Gregor et al, DRAW: A Recurrent Neural Network For Image Generation, ICML 2015

DRAW: Deep Recurrent Attentive Writer[5]



Learnt Parameters

$$(\tilde{g}_X, \tilde{g}_Y, \log\sigma^2, \log\tilde{\delta}, \log\gamma) = W(h^{dec})$$

$$g_X = \frac{A+1}{2}(\tilde{g}_X + 1)$$

$$g_Y = \frac{B+1}{2}(\tilde{g}_Y + 1)$$

$$\delta = \frac{\max(A,B)-1}{N-1}\tilde{\delta}$$

Inferred Parameters/Filters

$$\mu_X^i = g_X + (i - N/2 - 0.5)\,\delta$$

$$\mu_Y^j = g_Y + (j - N/2 - 0.5)\,\delta$$

[5] Gregor et al, DRAW: A Recurrent Neural Network For Image Generation, ICML 2015

And then came...

...self-attention! Largely inspired by the need in machine translation

And then came...

...self-attention! Largely inspired by the need in machine translation

- Encoder-becoder DNN models (RNNs, LSTMs, GRUs) were used for neural machine translation then
  - Encoder takes input sentence and creates a summary (last hidden layer also called *Context Vector*).
  - Decoder translates input sentence sequentially by processing the summary.

And then came...

...self-attention! Largely inspired by the need in machine translation

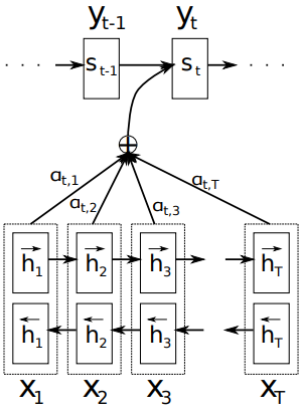
- Encoder-becoder DNN models (RNNs, LSTMs, GRUs) were used for neural machine translation then
  - Encoder takes input sentence and creates a summary (last hidden layer also called *Context Vector*).
  - Decoder translates input sentence sequentially by processing the summary.
- Problems with this approach:
  - Translation quality depends on quality of summary
  - RNNs/LSTMs create bad summaries for longer sentences (long-range dependency problem).
  - We can't give more importance to a set of words compared to others in the input sentence

And then came...

...self-attention! Largely inspired by the need in machine translation

- Encoder-becoder DNN models (RNNs, LSTMs, GRUs) were used for neural machine translation then
  - Encoder takes input sentence and creates a summary (last hidden layer also called *Context Vector*).
  - Decoder translates input sentence sequentially by processing the summary.
- Problems with this approach:
  - Translation quality depends on quality of summary
  - RNNs/LSTMs create bad summaries for longer sentences (long-range dependency problem).
  - We can't give more importance to a set of words compared to others in the input sentence
- Attention was proposed as a solution to all these problems.

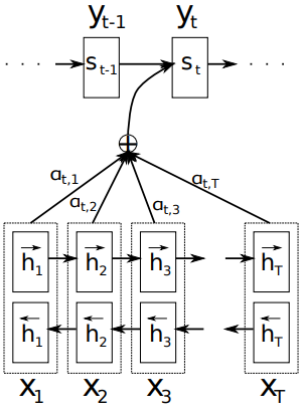
Origin of Self-Attention and Transformers[6]



- Introduced by Bahdanau et al to improve the encoder-decoder based neural machine translation in NLP

[6] Bahdanau et al, Neural Machine Translation by Jointly Learning to Align and Translate, ICLR 2015

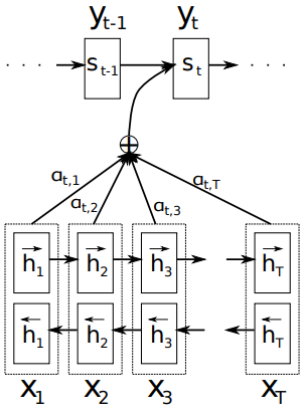
Origin of Self-Attention and Transformers[6]



- Introduced by Bahdanau et al to improve the encoder-decoder based neural machine translation in NLP
- They use a BiRNN as encoder and generate annotation h_j by concatenating forward and backward hidden states

[6] Bahdanau et al, Neural Machine Translation by Jointly Learning to Align and Translate, ICLR 2015

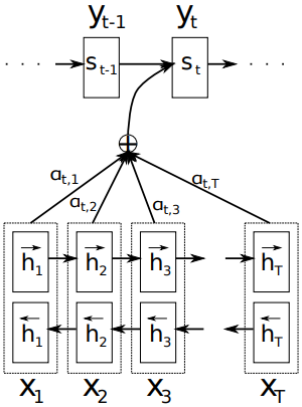
Origin of Self-Attention and Transformers[6]



- Introduced by Bahdanau et al to improve the encoder-decoder based neural machine translation in NLP
- They use a BiRNN as encoder and generate annotation h_j by concatenating forward and backward hidden states
- Each annotation h_j contains information about the whole input sequence with more emphasis around the j-th word.

[6] Bahdanau et al, Neural Machine Translation by Jointly Learning to Align and Translate, ICLR 2015

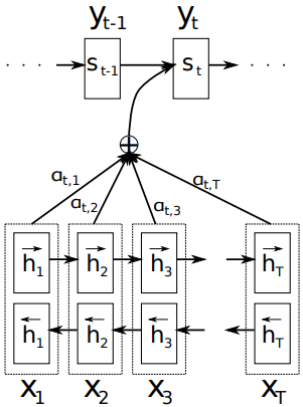
Origin of Self-Attention and Transformers[6]



- Introduced by Bahdanau et al to improve the encoder-decoder based neural machine translation in NLP
- They use a BiRNN as encoder and generate annotation h_j by concatenating forward and backward hidden states
- Each annotation h_j contains information about the whole input sequence with more emphasis around the j-th word.
- To focus on all input word embeddings while creating a context vector, they use a weighted sum of the hidden states rather than the final hidden state

[6] Bahdanau et al, Neural Machine Translation by Jointly Learning to Align and Translate, ICLR 2015

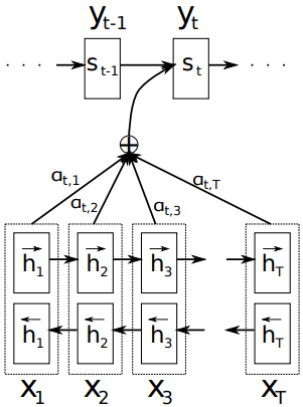
Origin of Self-Attention and Transformers



- For output word y_i, context vector c_i is the weighted sum of annotations h_j:

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

Origin of Self-Attention and Transformers



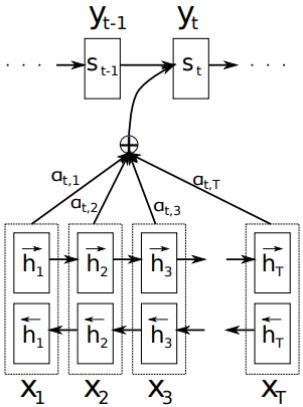
- For output word y_i, context vector c_i is the weighted sum of annotations h_j:

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

- The weight α_{ij} for a given annotation h_j is:

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

Origin of Self-Attention and Transformers



- For output word y_i, context vector c_i is the weighted sum of annotations h_j:

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

- The weight α_{ij} for a given annotation h_j is:

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

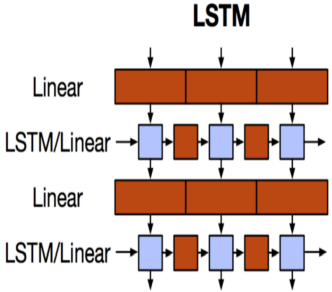
where

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

is an $alignment\ model$ that finds how well the j-th input matches with the i-th output.

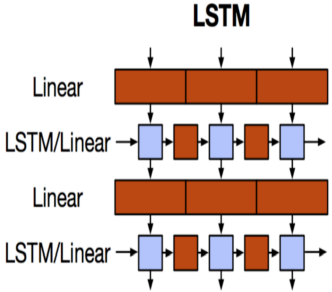
Motivation for Transformers

- Sequential computation prevents parallelization



Motivation for Transformers

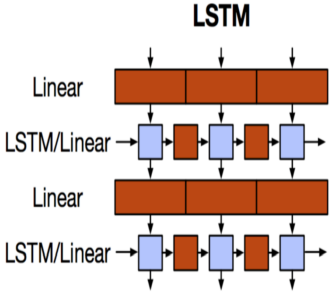
- Sequential computation prevents parallelization



- 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

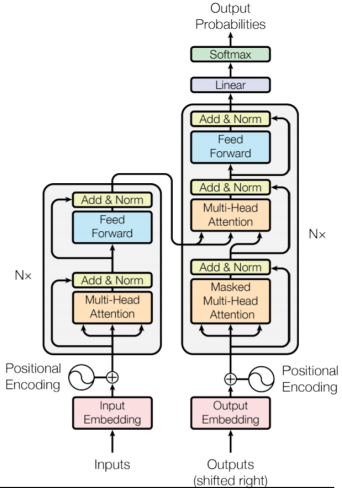
Motivation for Transformers

- 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?!

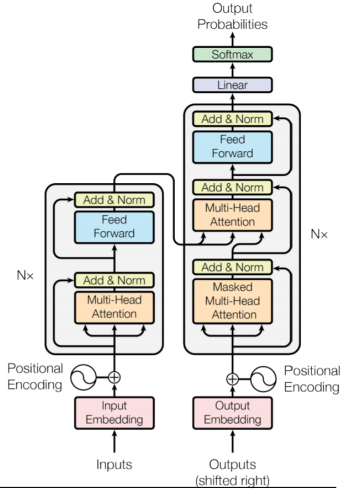
Transformers[7]



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

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

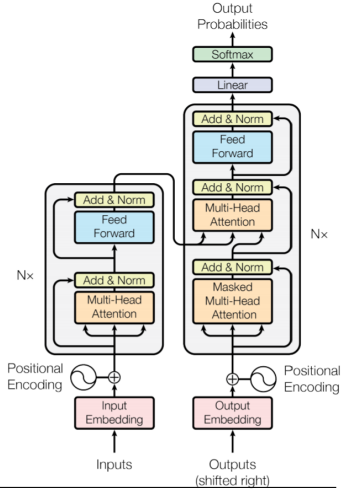
Transformers[7]



- 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

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

Transformers[7]



- 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

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

A Detour: The X-is-all-you-need Fascination

- One epoch is all you need, 2019

A Detour: The X-is-all-you-need Fascination

- One epoch is all you need, 2019
- Segmentation is All You Need, 2019

A Detour: The X-is-all-you-need Fascination

- One epoch is all you need, 2019
- Segmentation is All You Need, 2019
- Logarithmic pruning is all you need, 2020

A Detour: The X-is-all-you-need Fascination

- One epoch is all you need, 2019
- Segmentation is All You Need, 2019
- Logarithmic pruning is all you need, 2020
- Hopfield networks is all you need, 2020

A Detour: The X-is-all-you-need Fascination

- One epoch is all you need, 2019
- Segmentation is All You Need, 2019
- Logarithmic pruning is all you need, 2020
- Hopfield networks is all you need, 2020
- 15 Keypoints Is All You Need, 2020

A Detour: The X-is-all-you-need Fascination

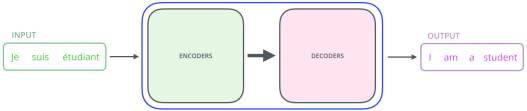
- One epoch is all you need, 2019
- Segmentation is All You Need, 2019
- Logarithmic pruning is all you need, 2020
- Hopfield networks is all you need, 2020
- 15 Keypoints Is All You Need, 2020

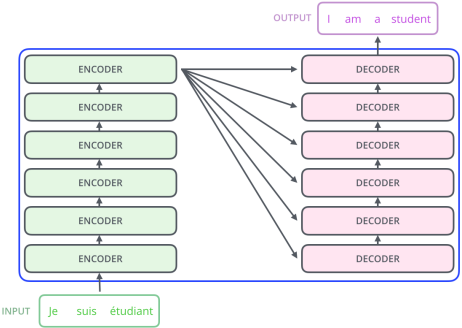
For more, please see `https://github.com/vinayprabhu/X-is-all-you-need`

Transformers in a Nutshell

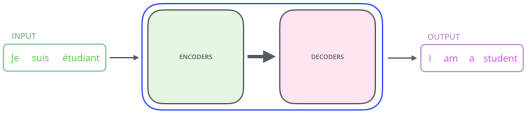
Transformers in a Nutshell

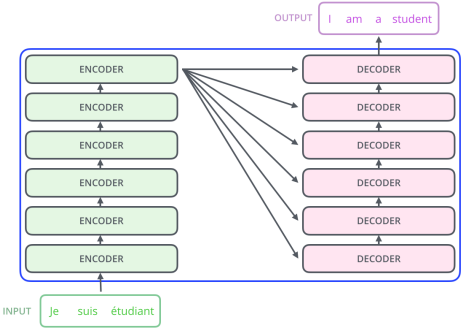
Transformers in a Nutshell

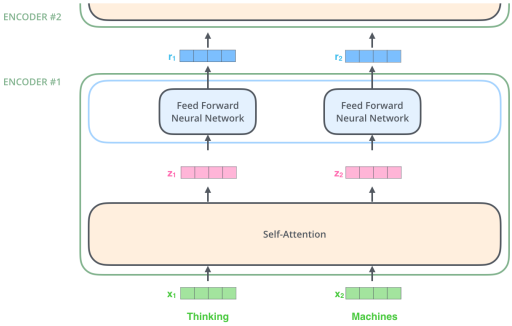




Transformers in a Nutshell

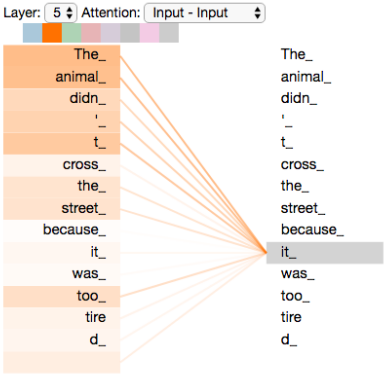




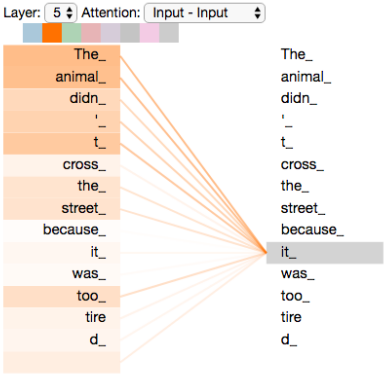


Self-Attention

- Consider two input sentences we want to translate:

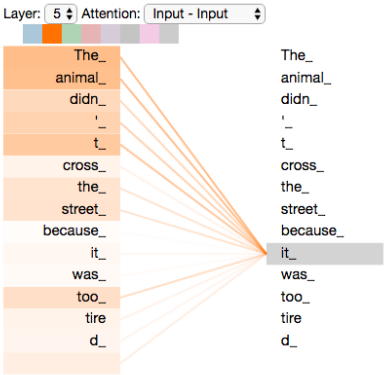


Self-Attention



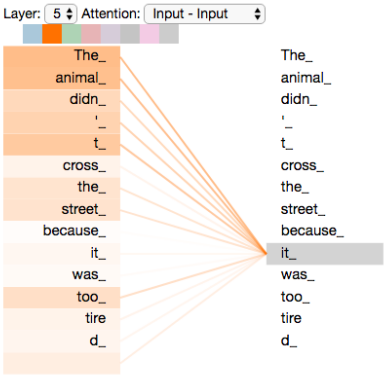
- Consider two input sentences we want to translate:
  - *The animal didn't cross the street because it was too tired*

Self-Attention



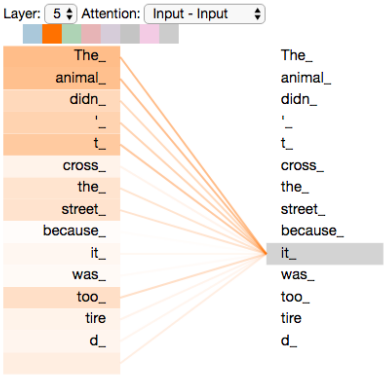
- Consider two input sentences we want to translate:
  - *The animal didn't cross the street because it was too tired*
  - *The animal didn't cross the street because it was too wide*

Self-Attention



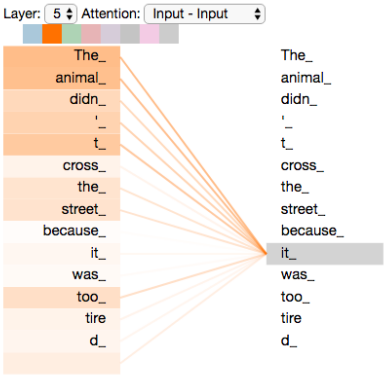
- Consider two input sentences we want to translate:
  - *The animal didn't cross the street because it was too tired*
  - *The animal didn't cross the street because it was too wide*
- "it" refers to "animal" in first case, but to "street" in second case; this is hard for traditional Seq2Seq models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

Self-Attention



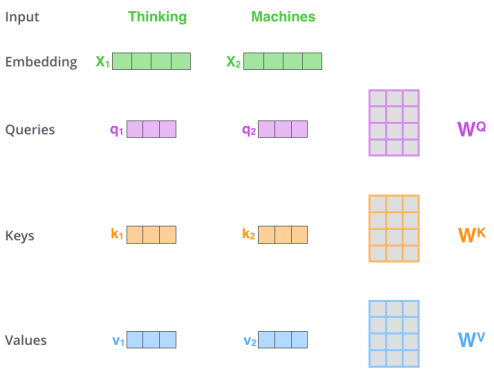
- Consider two input sentences we want to translate:
  - *The animal didn't cross the street because it was too tired*
  - *The animal didn't cross the street because it was too wide*
- "it" refers to "animal" in first case, but to "street" in second case; this is hard for traditional Seq2Seq models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



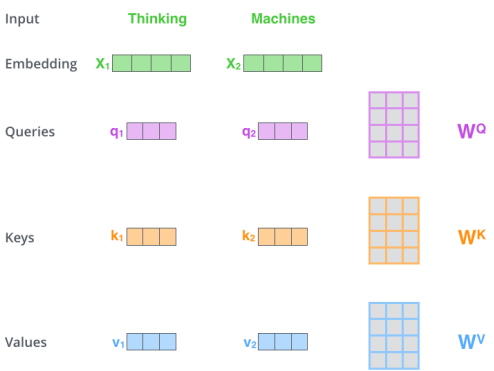
- Consider two input sentences we want to translate:
  - *The animal didn't cross the street because it was too tired*
  - *The animal didn't cross the street because it was too wide*
- "it" refers to "animal" in first case, but to "street" in second case; this is hard for traditional Seq2Seq models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



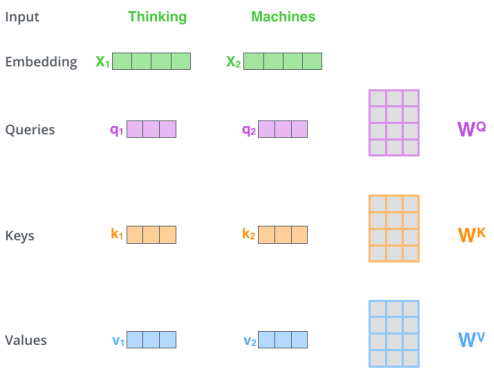
- **STEP 1:** Create three vectors from encoder's input vector (x_i):
  - Query vector (q_i)
  - Key vector (k_i)
  - Value vector (v_i)

Self-Attention



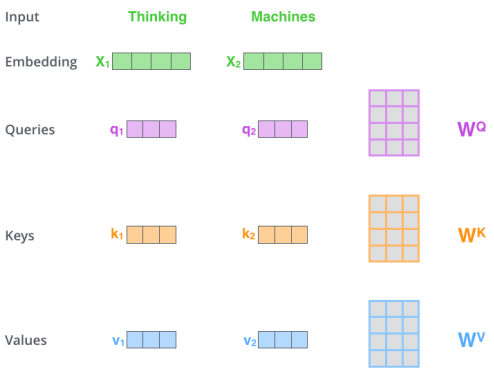
- **STEP 1:** Create three vectors from encoder's input vector (x_i):
  - Query vector (q_i)
  - Key vector (k_i)
  - Value vector (v_i)
- These are created by multiplying input with weight matrices W^Q, W^K, W^V, learned during training

Self-Attention



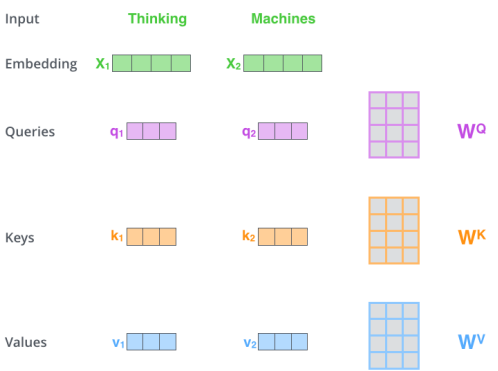
- **STEP 1:** Create three vectors from encoder's input vector (x_i):
  - Query vector (q_i)
  - Key vector (k_i)
  - Value vector (v_i)
- These are created by multiplying input with weight matrices W^Q, W^K, W^V, learned during training
- In the paper, $q, k, v \in \mathbb{R}^{64}$ and $x \in \mathbb{R}^{512}$

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?

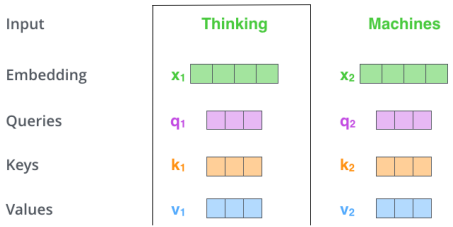
Self-Attention



- **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?

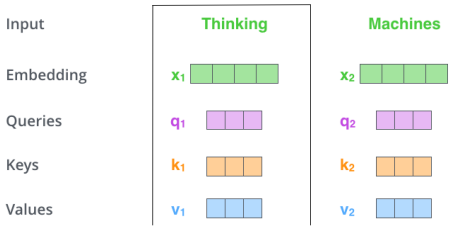
Self-Attention

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



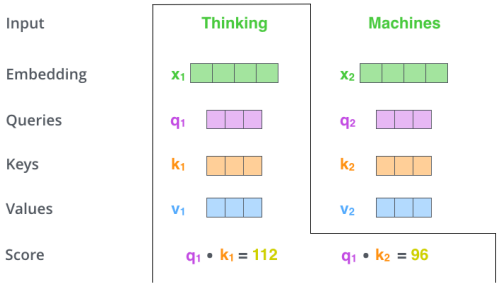
Self-Attention

- **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



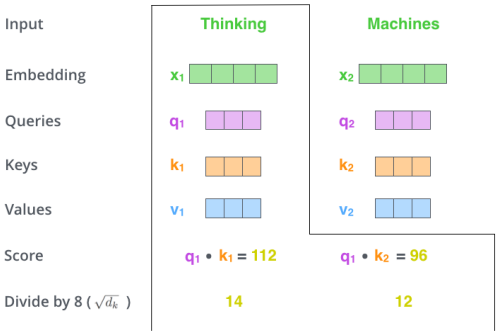
Self-Attention

- **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



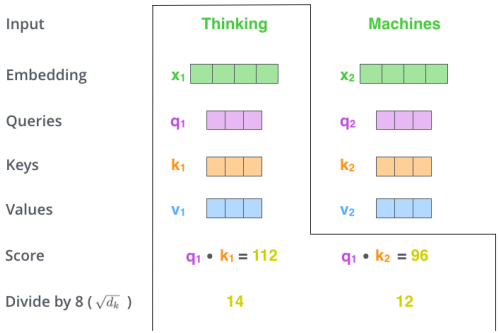
Self-Attention

- **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 $\sqrt{length(k)}$



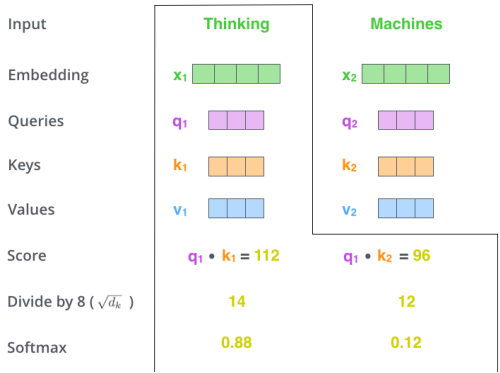
Self-Attention

- **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**, this design choice leads to more stable gradients



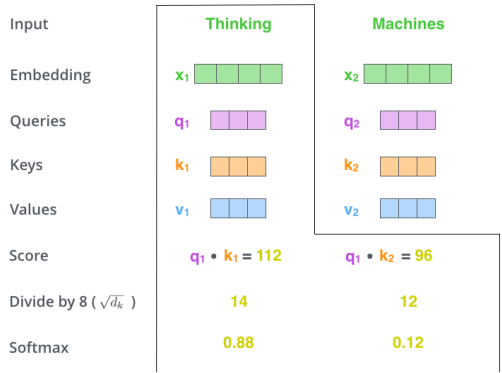
Self-Attention

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



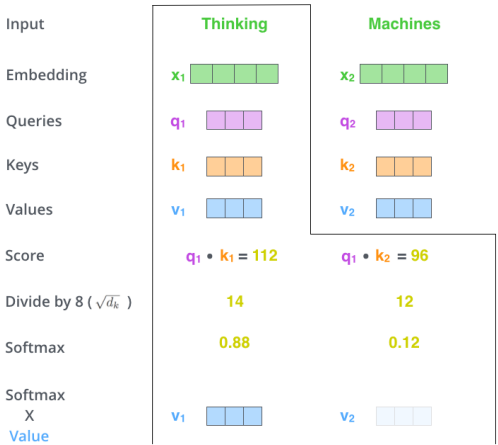
Self-Attention

- **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



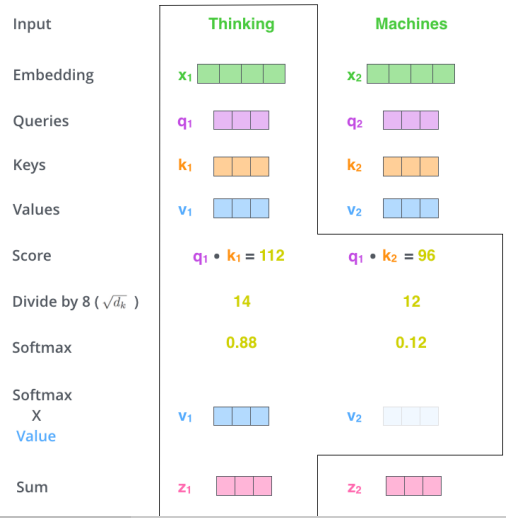
Self-Attention

- **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

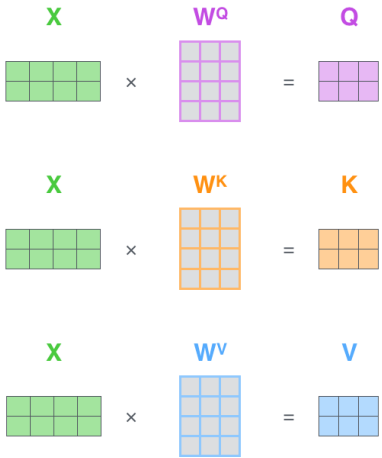


Self-Attention

- **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)

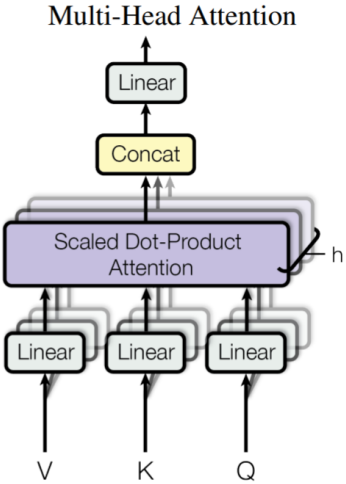


Self-Attention: Illustration



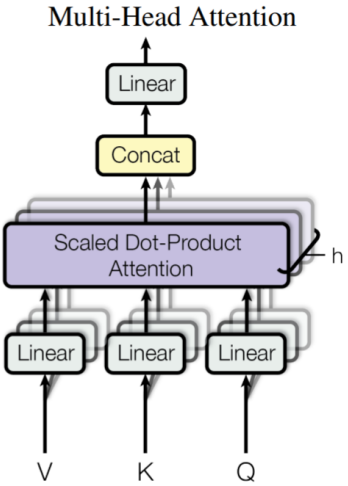
$$\text{softmax}\left(\frac{Q \times K^T}{\sqrt{d_k}}\right) V = Z$$

Multi-Head Attention



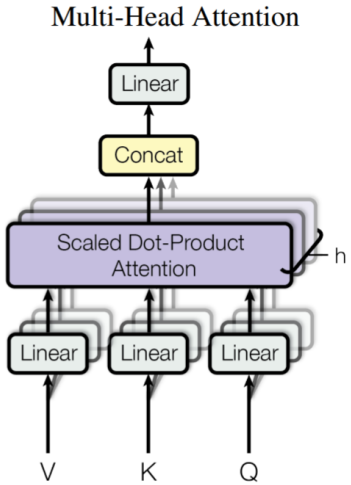
- Improves performance of the attention layer in two ways:

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

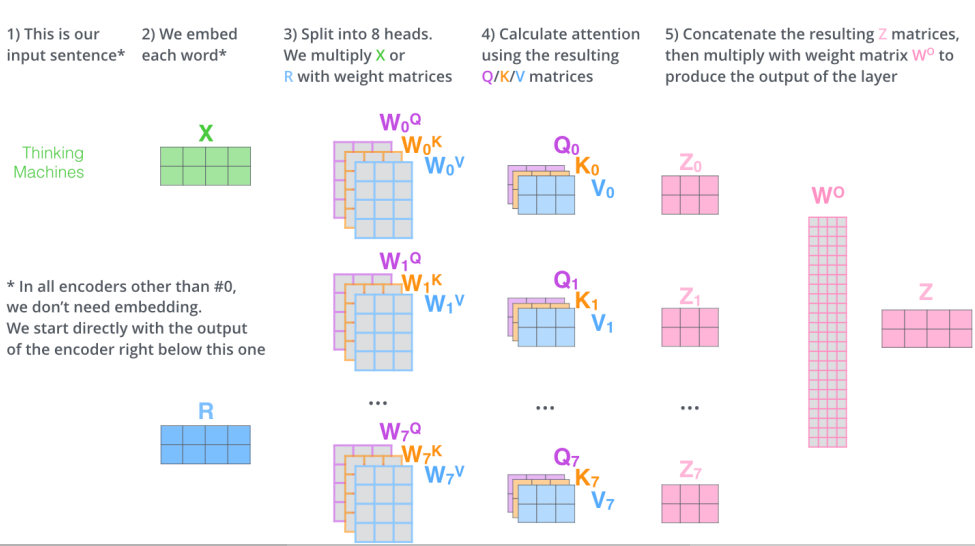
Multi-Head Attention



- 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

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

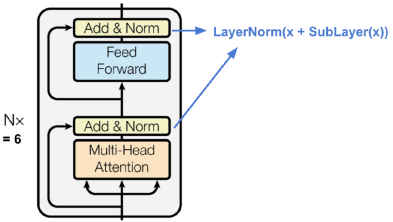
Positional Encoding

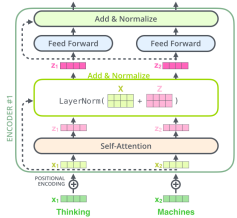
- Unlike RNN and CNN encoders, attention encoder outputs do not depend on order of inputs (Why?)
- But order of sequence conveys important information for machine translation tasks and language modeling
- The idea: Add positional information of input token in the sequence into input embedding vectors

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

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

Encoder

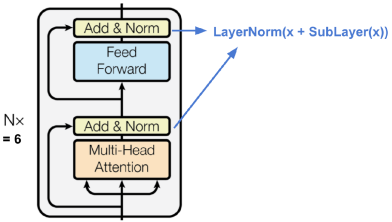


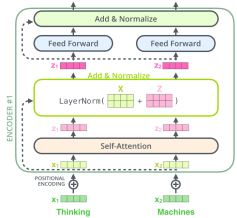


- Stack of N=6 identical layers

Credit: "Attention? Attention!" by Lilian Weng

Encoder

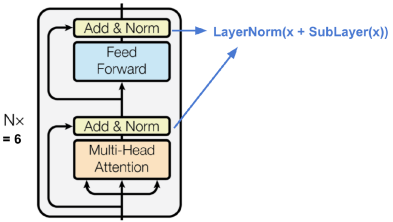


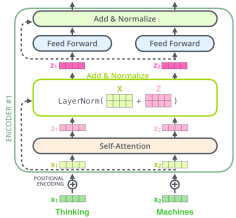


- Stack of N=6 identical layers
- Each layer has a **multi-head self-attention layer** and a simple position-wise fully connected **feedforward network**

Credit: "Attention? Attention!" by Lilian Weng

Encoder

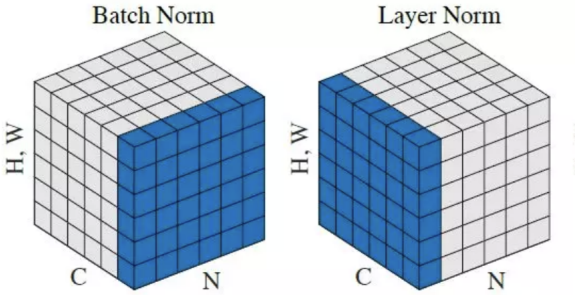




- Stack of N=6 identical layers
- Each layer has a **multi-head self-attention layer** and a simple position-wise fully connected **feedforward network**
- Each sub-layer has a **residual** connection and **layer-normalization**; all sub-layers output data of same dimension $d_{model} = 512$

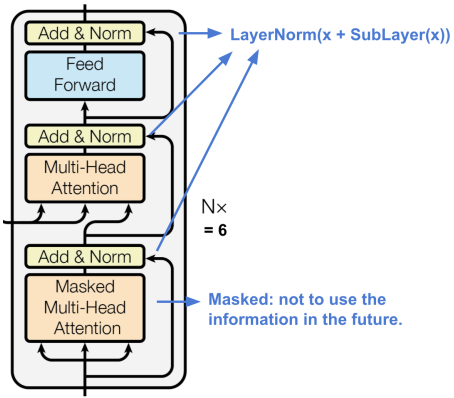
Credit: "Attention? Attention!" by Lilian Weng

Layer Normalization[8]



[8] Ba, Kiros, Hinton, Layer Normalization, 2016

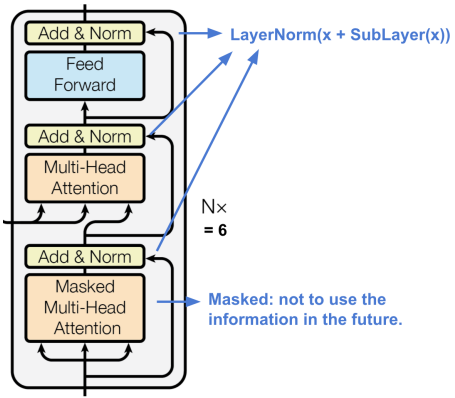
Decoder



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

Credit: "Attention? Attention!" by Lilian Weng

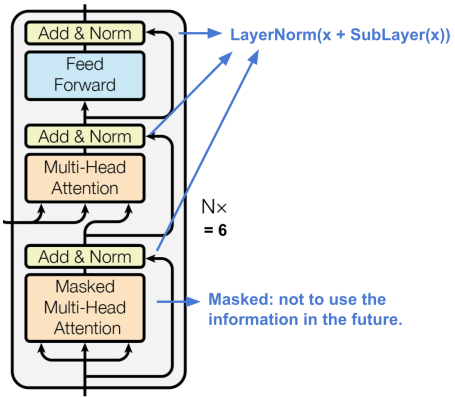
Decoder



- 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

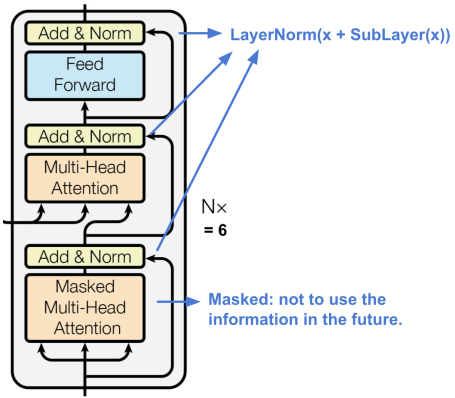
Decoder



- 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

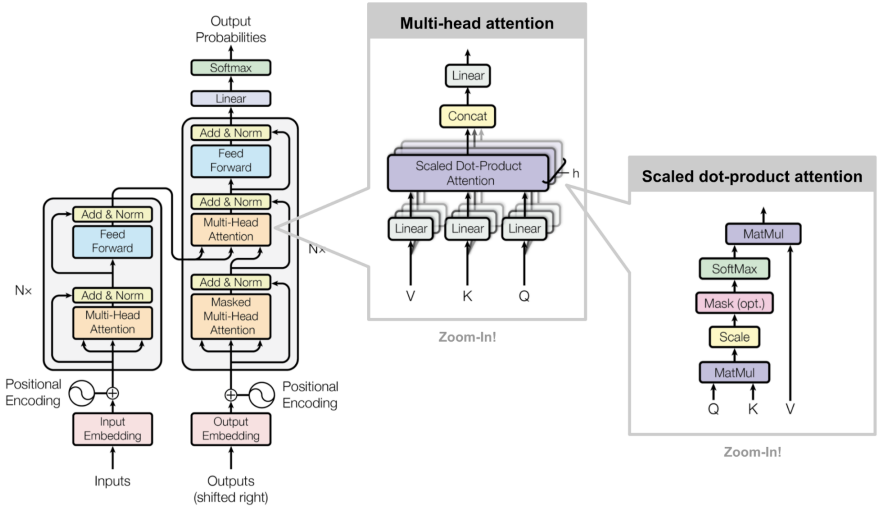
Decoder



- 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

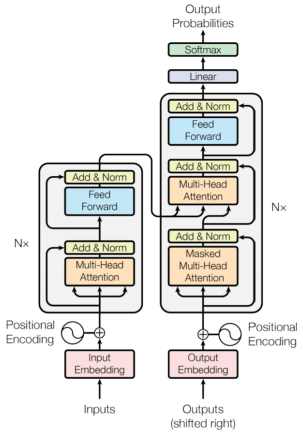
Credit: "Attention? Attention!" by Lilian Weng

Transformers: Full Architecture



Credit: "Attention? Attention!" by Lilian Weng

Transformers in NLP: Neural Machine Translation [9]

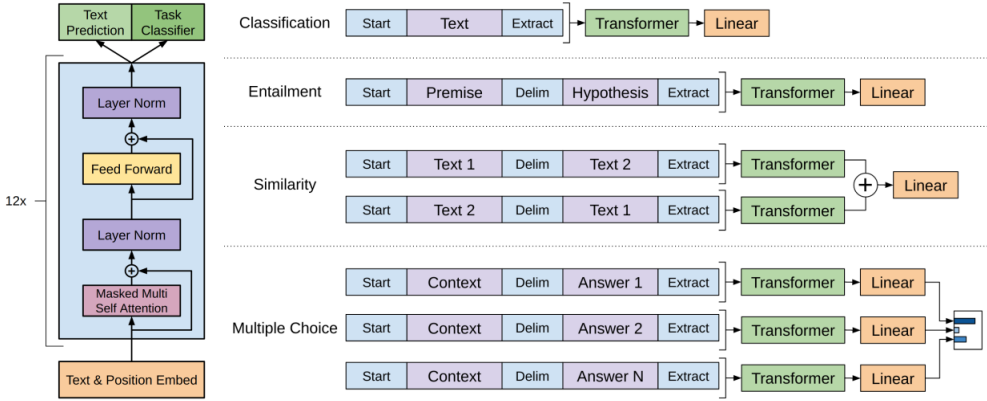


Results on English-to-German and English-to-French newstest2014 tests

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**	$2.3 \cdot 10^{19}$	

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

Transformers in NLP: Language Understanding [10]



[10] Radford et al, Improving Language Understanding by Generative Pre-Training, Technical report, OpenAI

Transformers in NLP: Language Understanding

- Pre-training language models showed impressive results on multiple natural language processing tasks like natural language inference, paraphrasing, named entity recognition and question answering.

[11] Peters et al, Deep contextualized word representations, NAACL 2018.

[12] Radford et al, Improving Language Understanding by Generative Pre-Training, Technical report, OpenAI

Transformers in NLP: Language Understanding

- Pre-training language models showed impressive results on multiple natural language processing tasks like natural language inference, paraphrasing, named entity recognition and question answering.
- To apply the learnt language representations to downstream tasks, two strategies are followed: $feature-based$ and $fine-tuning$.
  - **Feature-based**[11] methods develop task specific architectures and utilize the pre-trained knowledge in solving the task
  - **Fine-tuning**[12] approaches try to minimize task-specific parameters and solve the task by fine-tuning pre-trained parameters

[11] Peters et al, Deep contextualized word representations, NAACL 2018.

[12] Radford et al, Improving Language Understanding by Generative Pre-Training, Technical report, OpenAI

Transformers in NLP: Language Understanding

- Pre-training language models showed impressive results on multiple natural language processing tasks like natural language inference, paraphrasing, named entity recognition and question answering.
- To apply the learnt language representations to downstream tasks, two strategies are followed: $feature-based$ and $fine-tuning$.
  - **Feature-based**[11] methods develop task specific architectures and utilize the pre-trained knowledge in solving the task
  - **Fine-tuning**[12] approaches try to minimize task-specific parameters and solve the task by fine-tuning pre-trained parameters
- Both these strategies used unidirectional language models for pre-training which restricted the power of the language representations for fine-tuning.

[11] Peters et al, Deep contextualized word representations, NAACL 2018.

[12] Radford et al, Improving Language Understanding by Generative Pre-Training, Technical report, OpenAI

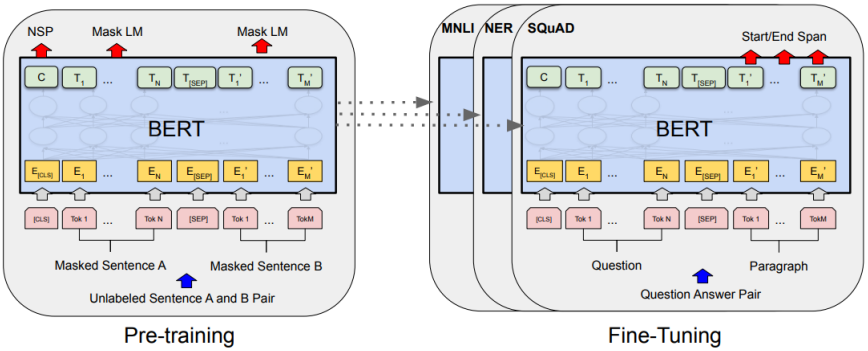
Transformers in NLP: Language Understanding

- Pre-training language models showed impressive results on multiple natural language processing tasks like natural language inference, paraphrasing, named entity recognition and question answering.
- To apply the learnt language representations to downstream tasks, two strategies are followed: $feature-based$ and $fine-tuning$.
  - **Feature-based**[11] methods develop task specific architectures and utilize the pre-trained knowledge in solving the task
  - **Fine-tuning**[12] approaches try to minimize task-specific parameters and solve the task by fine-tuning pre-trained parameters
- Both these strategies used unidirectional language models for pre-training which restricted the power of the language representations for fine-tuning.
- Jacob et al solved this problem by using a 'masked language model' (MLM) pre-training objective in their paper, BERT.

[11] Peters et al, Deep contextualized word representations, NAACL 2018.

[12] Radford et al, Improving Language Understanding by Generative Pre-Training, Technical report, OpenAI

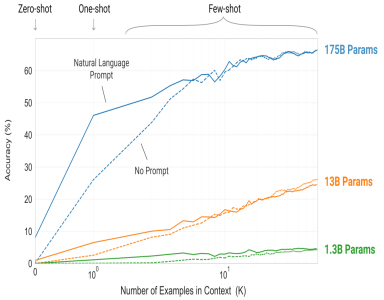
Transformers in NLP: Language Understanding[13]



[13] Jacob et al, BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding , NAACL 2019

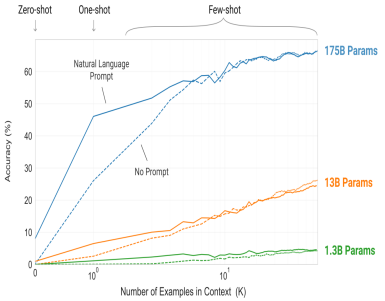
Transformers in NLP: Language Understanding[14]

- Recent works achieved performance gains on many NLP tasks by pre-training on large data and fine-tuning on specific tasks



[14] Brown et al, Language models are few-shot learners, NeurIPS 2020.

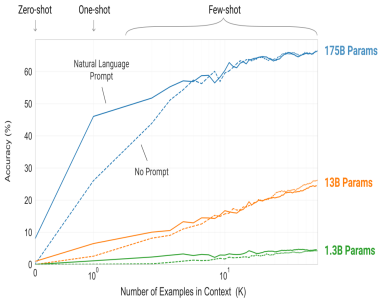
Transformers in NLP: Language Understanding[14]



- Recent works achieved performance gains on many NLP tasks by pre-training on large data and fine-tuning on specific tasks
- Although task-agnostic, these methods still require large task-specific data for fine-tuning. In contrary, humans can learn a new language task by simple instructions only.

[14] Brown et al, Language models are few-shot learners, NeurIPS 2020.

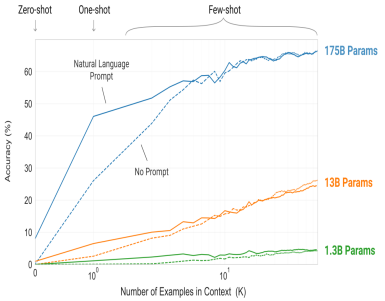
Transformers in NLP: Language Understanding[14]



- Recent works achieved performance gains on many NLP tasks by pre-training on large data and fine-tuning on specific tasks
- Although task-agnostic, these methods still require large task-specific data for fine-tuning. In contrary, humans can learn a new language task by simple instructions only.
- In an attempt to bridge this gap, Brown et al show that we can improve task-agnostic few-shot performance by scaling up language models

[14] Brown et al, Language models are few-shot learners, NeurIPS 2020.

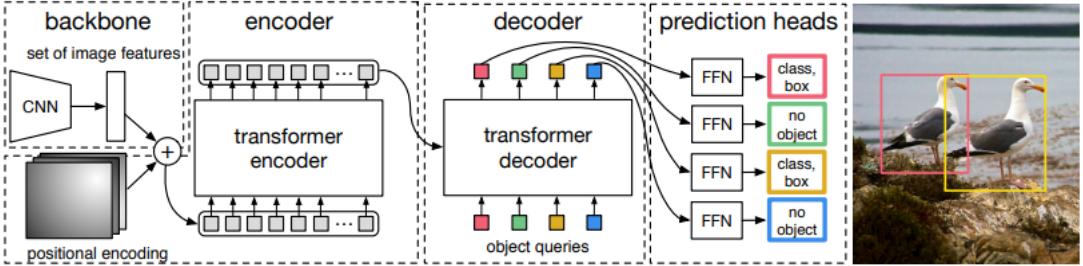
Transformers in NLP: Language Understanding[14]



- Recent works achieved performance gains on many NLP tasks by pre-training on large data and fine-tuning on specific tasks
- Although task-agnostic, these methods still require large task-specific data for fine-tuning. In contrary, humans can learn a new language task by simple instructions only.
- In an attempt to bridge this gap, Brown et al show that we can improve task-agnostic few-shot performance by scaling up language models
- Their model, named GPT-3 is trained with 175 billion parameters and it shows significant performance gain when tested in few-shot setting.

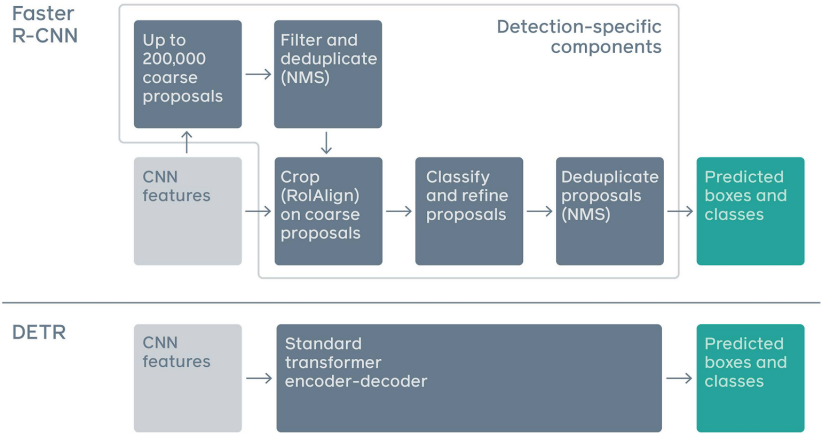
[14] Brown et al, Language models are few-shot learners, NeurIPS 2020.

Transformers in Computer Vision: Object Detection[15]



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

Transformers in Computer Vision: Object Detection



Credit: Ram Sagar, Analytics India Mag

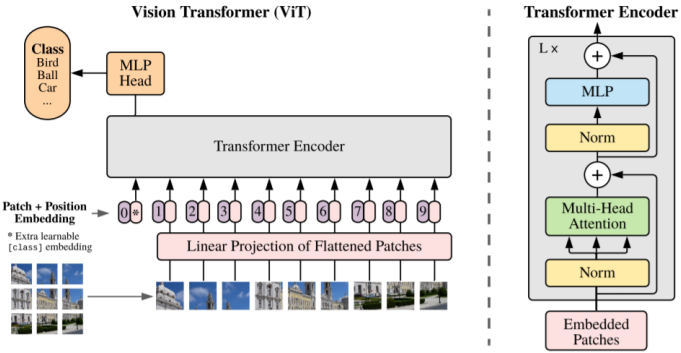
Transformers in Computer Vision: Object Detection[16]

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**

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

Transformers in Computer Vision: Image Recognition[17]



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

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

Transformers in Computer Vision: General Purpose Backbone[18]

- Problems in the previous transformer architectures:
  - Tokens are of fixed scale in NLP problems, which is not suitable in vision tasks.
  - Computational complexity is quadratic to image size in ViT model and using high resolution images for dense pixel level prediction tasks would make the computation intractable.

[18] Liu, Ze, et al, Swin transformer: Hierarchical vision transformer using shifted windows, CVPR 2021.

Transformers in Computer Vision: General Purpose Backbone[18]

- Problems in the previous transformer architectures:
  - Tokens are of fixed scale in NLP problems, which is not suitable in vision tasks.
  - Computational complexity is quadratic to image size in ViT model and using high resolution images for dense pixel level prediction tasks would make the computation intractable.
- To solve these problems, Liu et al proposed Swin Transformer with hierarchical feature maps and linear computational complexity.

[18] Liu, Ze, et al, Swin transformer: Hierarchical vision transformer using shifted windows, CVPR 2021.

Transformers in Computer Vision: General Purpose Backbone[18]

- Problems in the previous transformer architectures:
  - Tokens are of fixed scale in NLP problems, which is not suitable in vision tasks.
  - Computational complexity is quadratic to image size in ViT model and using high resolution images for dense pixel level prediction tasks would make the computation intractable.
- To solve these problems, Liu et al proposed Swin Transformer with hierarchical feature maps and linear computational complexity.
- An image is split into non-overlapping patches and to produce hierarchical feature maps, these patches are merged as we go deep into the network.

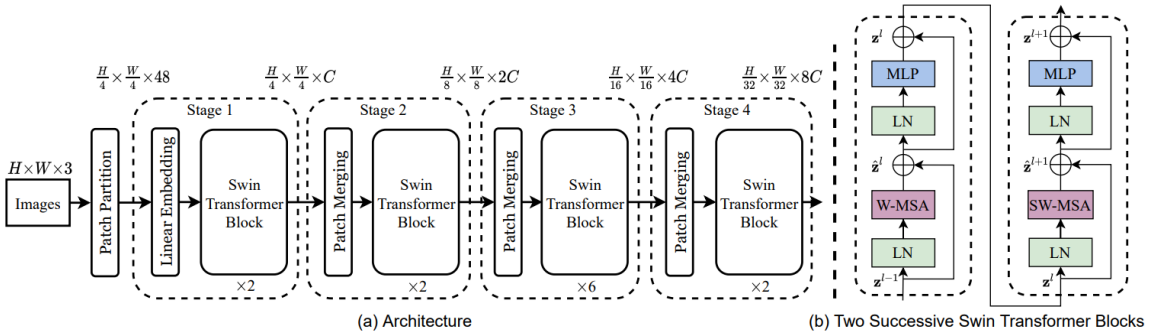
[18] Liu, Ze, et al, Swin transformer: Hierarchical vision transformer using shifted windows, CVPR 2021.

Transformers in Computer Vision: General Purpose Backbone[18]

- Problems in the previous transformer architectures:
  - Tokens are of fixed scale in NLP problems, which is not suitable in vision tasks.
  - Computational complexity is quadratic to image size in ViT model and using high resolution images for dense pixel level prediction tasks would make the computation intractable.
- To solve these problems, Liu et al proposed Swin Transformer with hierarchical feature maps and linear computational complexity.
- An image is split into non-overlapping patches and to produce hierarchical feature maps, these patches are merged as we go deep into the network.
- The original transformer block is replaced by a Swin transformer block that uses shifted windows and generates feature maps with a linear computational complexity.

[18] Liu, Ze, et al, Swin transformer: Hierarchical vision transformer using shifted windows, CVPR 2021.

Transformers in Computer Vision: General Purpose Backbone[19]



[19] Liu, Ze, et al, Swin transformer: Hierarchical vision transformer using shifted windows, CVPR 2021.

Transformers in other applications

- Transformers now used in various applications like image classification, object detection/tracking, speech recognition and graph representation.
- Dong et al proposed Speech-Transformer[20] that proposes a 2D-Attention mechanism that attends to both time and frequency axes jointly
- Liu et al introduced a self-supervised speech pre-training method called TERA[21] which uses alteration along three orthogonal axes to pre-train Transformer Encoders on a large amount of unlabeled speech

[20] L. Dong, S. Xu and B. Xu, Speech-Transformer: A No-Recurrence Sequence-to-Sequence Model for Speech Recognition, ICASSP 2018

[21] Liu et al, Tera: Self-supervised learning of transformer encoder representation for speech, ACM Transactions on Audio, Speech, and Language Processing 2021

Transformers in other applications

- Graph Neural Networks (GNNs) used to learn node representations on fixed and homogeneous graphs.Yun et al proposed Graph Transformer Networks[22] (GTNs) that are capable of learning from mis-specified and heterogeneous graphs.
- Transformers yet to top popular leader boards of graph level predictions as compared to the state-of-the-art GNN variants
- Ying et al proposed Graphormer[23], which attained excellent results on a broad range of graph representation learning tasks.

[22] Yun, Seongjun, et al, Graph transformer networks, NeurIPS 2019

[23] Ying, Chengxuan, et al, Do transformers really perform badly for graph representation?, NeurIPS 2021

Resources

- A Comprehensive Guide to Attention mechanism in Deep Learning by the american_express
- Most of the slides used for explaining the transformer are from The Illustrated Transformer article by Jay Alammar
- A detailed explanation of positional encoding by Amirhossein Kazemnejad
- An animated explanation for the Swin Transformer paper can be found here 1, 2.
- Speech Transformers article by Dmitry Obukhov.

And finally...

Why are transformers good stand-up comedy material?

- BERT, Albert, RoBERTa, ...

And finally...

Why are transformers good stand-up comedy material?

- BERT, Albert, RoBERTa, ...
- GPT, GPT-2, GPT-3,...

And finally...

Why are transformers good stand-up comedy material?

- BERT, Albert, RoBERTa, ...
- GPT, GPT-2, GPT-3,...
- Greekbert, Alberto, Camembert,....

And finally...

Why are transformers good stand-up comedy material?

- BERT, Albert, RoBERTa, ...
- GPT, GPT-2, GPT-3,...
- Greekbert, Alberto, Camembert,....
- The Evolved Transformer