Attention & Transformers
For Language and Vision

Adam Vinestock
Boaz Zimbler

Based on the papers

Attention is All you Need
An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale
Swin Transformer: Hierarchical Vision Transformer using Shifted Windows

Before we start....

Introduction

Time plan

Q & A

Table of Contents

01

Attention & Transformer Architecture

A brief history of deep architectures

Lex fridman
Ilya Sutskever

1994 — 2012 — 2014 — **2017**

LeNet5

One of the first CNN's Sequence of 3 layers: convolution, pooling, non-linearity

AlexNet

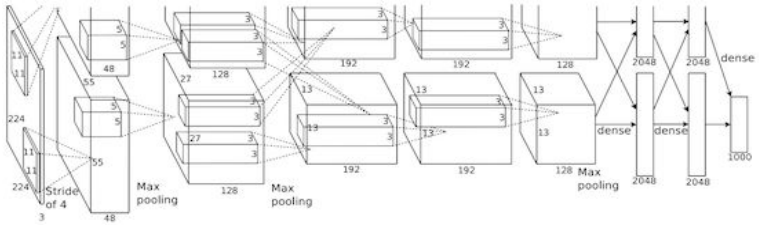
Scaled the insights of LeNet to much more complex hierarchies

LSTM

Became the state-of-the-art model for sequence modeling

Transformers

Introduced a new architecture for sequence modeling, replacing the RNN and CNN's in many tasks



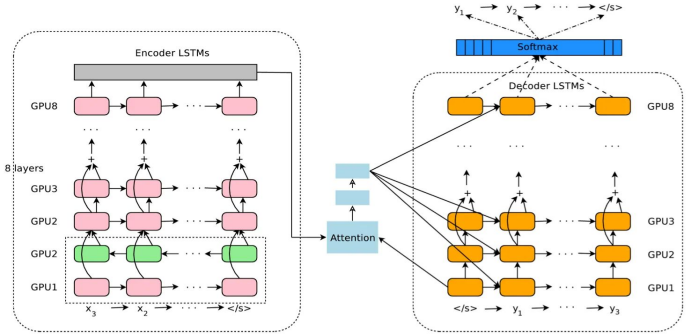
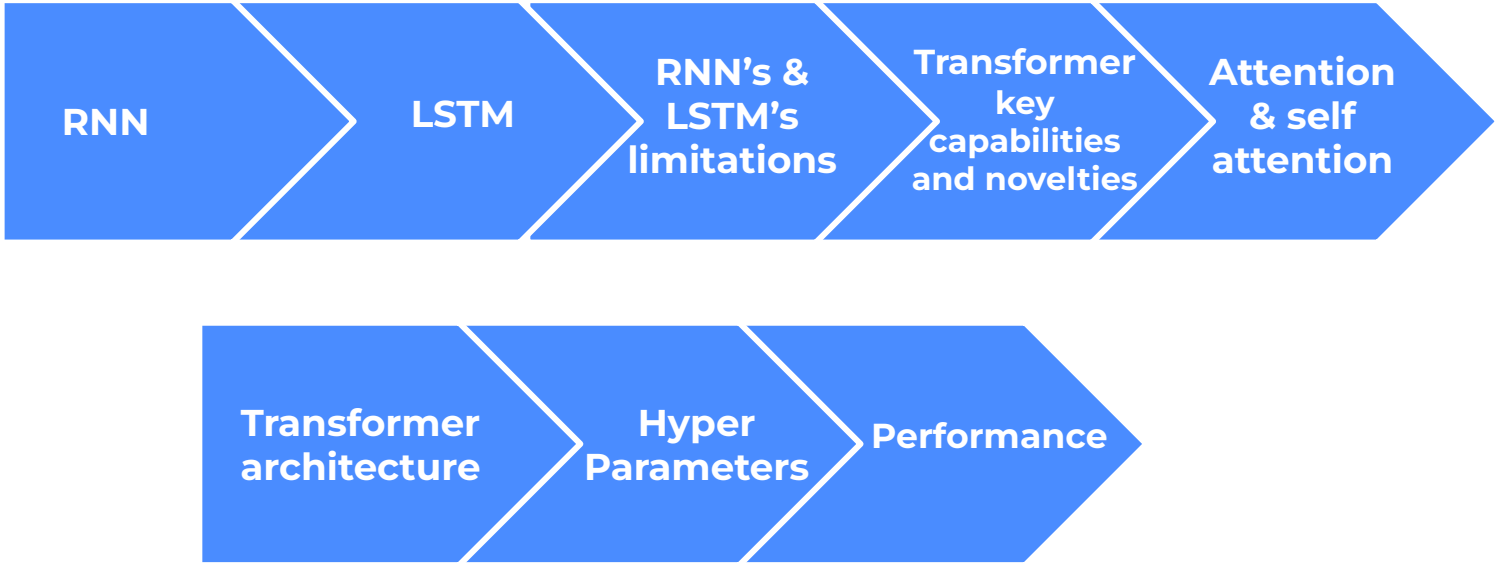


Image source: link1, link2

In This Chapter



Recurrent Neural Network

- Can process sequential data of variable length, such as time series, speech signals, text
- Works by iterating over a sequence of input vectors, using the output of each iteration as input to the next
- Maintains a hidden state that captures information about the previous inputs in the sequence (h_t)
- However - RNN's suffer from vanishing gradients!

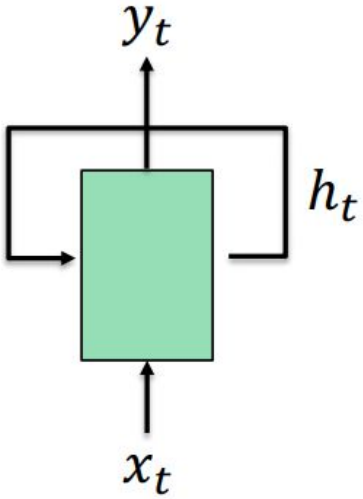


Image source: Rienchman 3598

Long Short-Term Memory

- Type of RNN that address the vanishing gradient problem by introducing gating mechanisms (input, forget, output)
- LSTM's selectively remember or forget information over time, making them better suited for modeling long-term dependencies
- But LSTM's are inherently sequential, which limits their ability to process data in parallel!

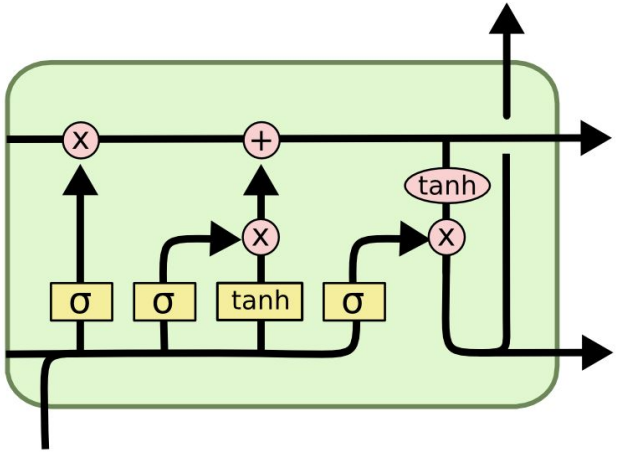


Image source: link

RNN and LSTM limitations

- RNN's & LSTM's have limitations in parallel processing since each step in the sequence depends on the previous step

- They can also be slow to train, especially on longer sequences, due to the need to backpropagate errors through the entire sequence

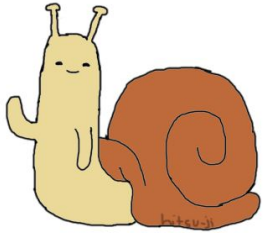
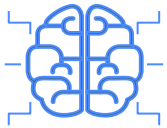


Image Source: link

Transformer

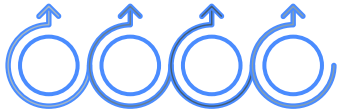
Parallel processing



Capturing long-range dependencies

Faster to train

Better accuracy

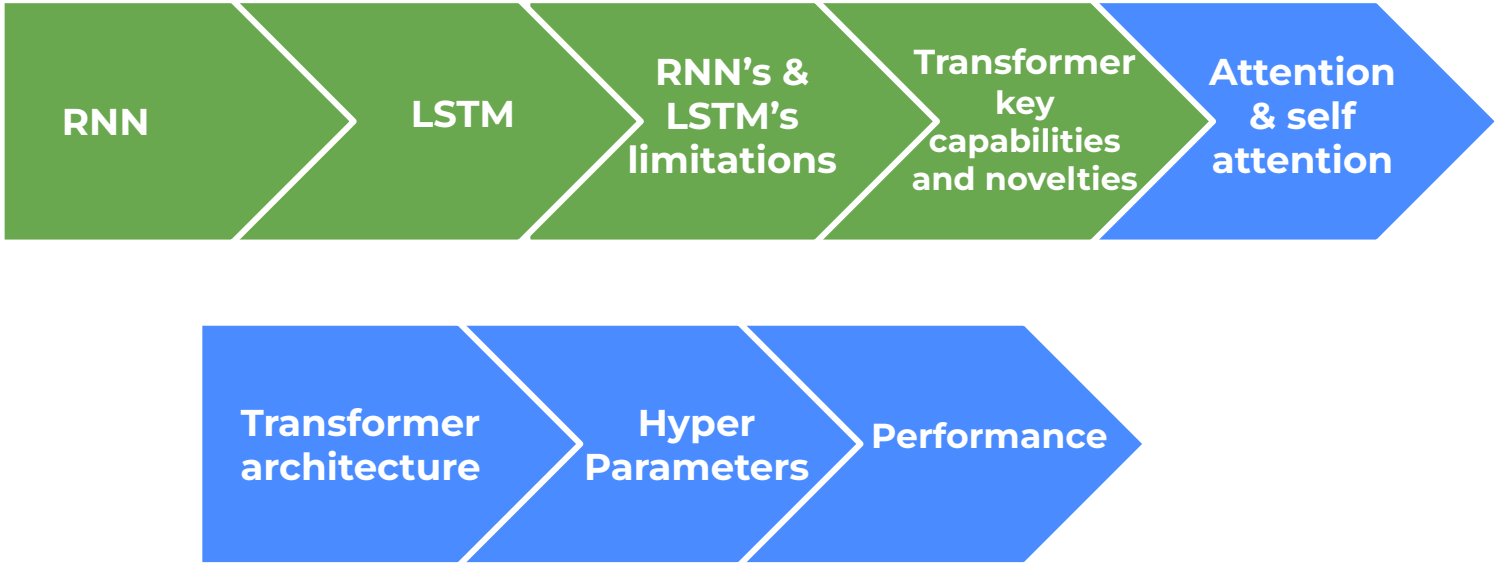


Inference is still semi recurrent
(Decoder is an Autoregressive model)

Transformer

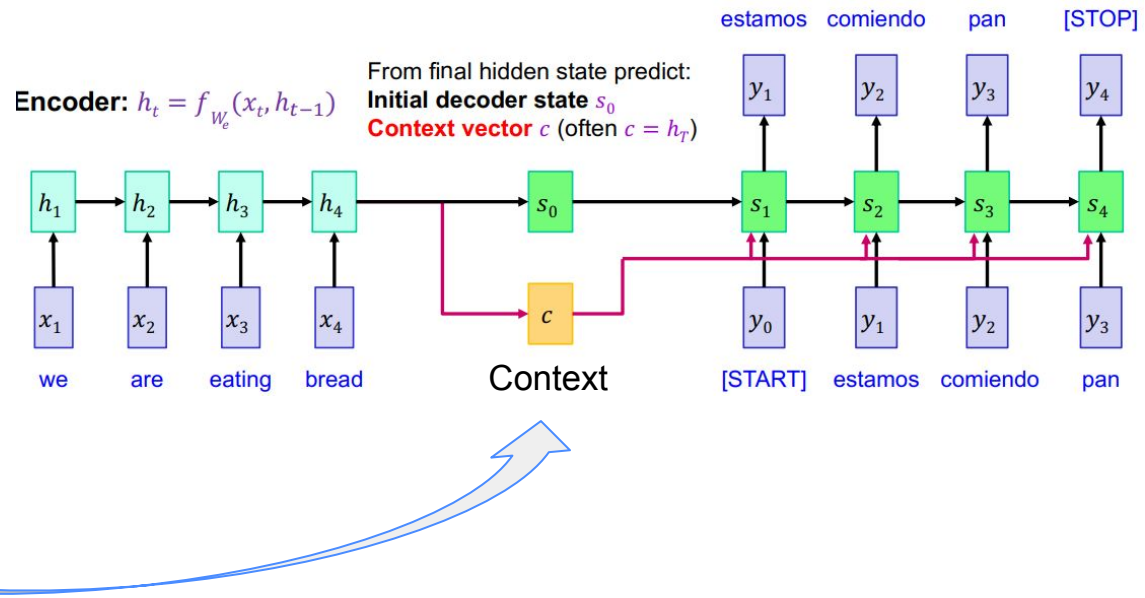
- First introduced in the paper "*Attention is all you need*" (2017)
- Key components and novelties
  - **Self-attention mechanism**: allows to attend to all positions in the input and output sequences, capturing long-range dependencies more effectively than RNN's or LSTM's
  - **Multi-head attention**: allows to jointly attend to information from different representation subspaces, improving attention-based features
  - **Positional encoding**: enables to incorporate the order of the input sequence into the attention computations, without requiring recurrent connections

In This Chapter



Attention Mechanism

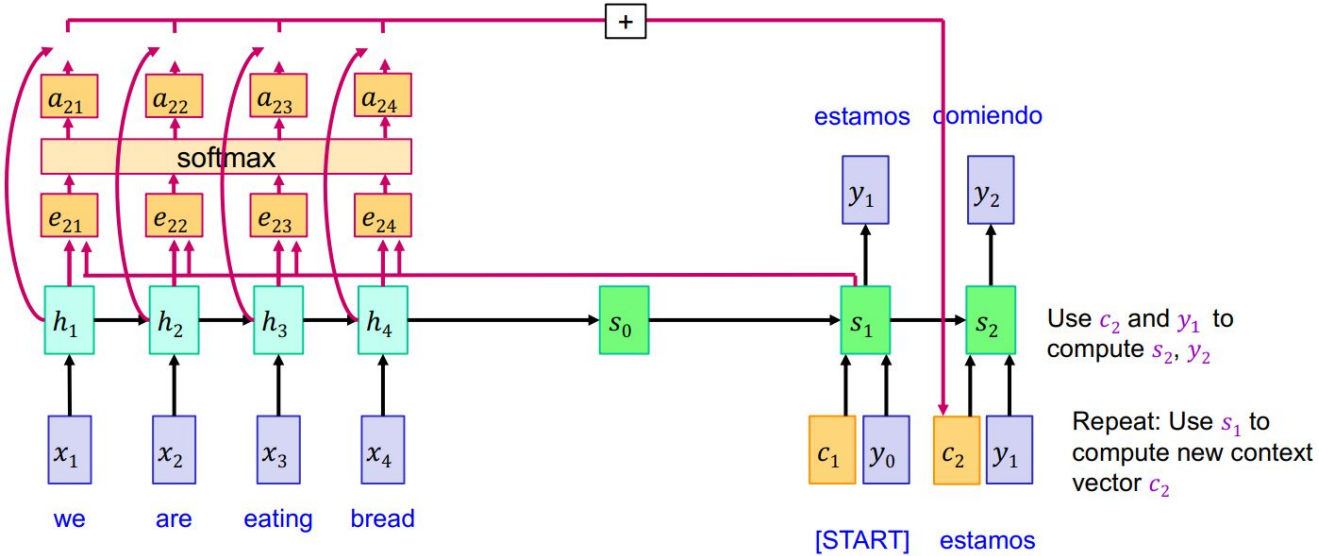
- Loss of information from the input sequence in decoder stage
- Perform a type of “skip connect” to attend over the encoder sequence
- A single context for all decoding steps



Adapted from: Rienchman 3598

Attention Mechanism

- Idea - use adaptive context vector at each step of the decoder
- Attention - At each timestep of decoder, context vector “looks at” different parts of the input sequence



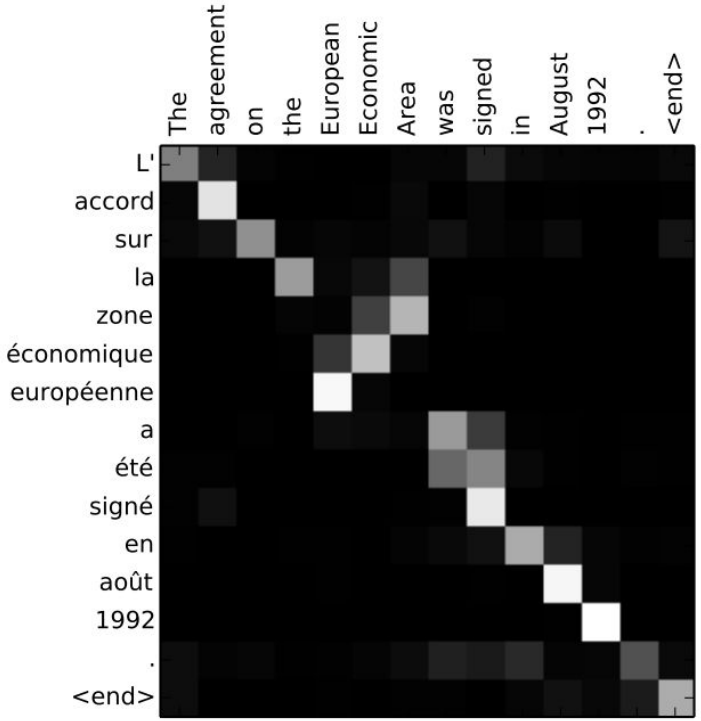
Adapted from: Rienchman 3598

Attention Mechanism

Video Source: Jay alammar

Attention Mechanism - example

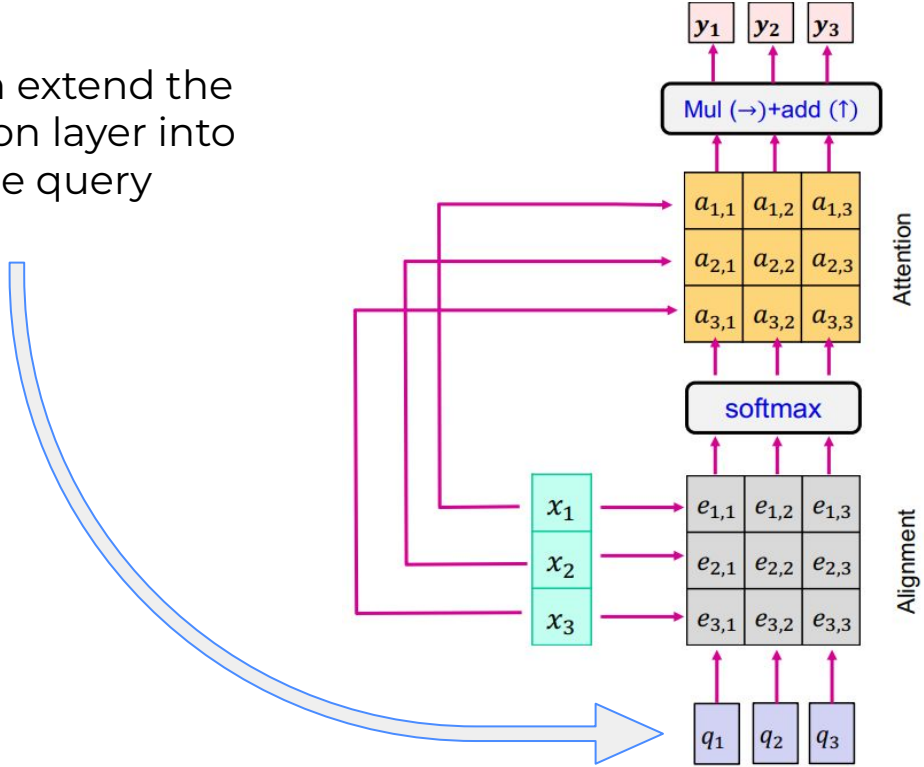
- Seq2Seq translation task
- Note - the model isn’t just mindlessly aligning the first word at the output with the first word from the input
- It actually learned from the training phase how to align words in that language pair (French and English in this example)



Adapted from: Jay alammar

General Attention layer

- We can extend the attention layer into multiple query vectors



Outputs:

$\boldsymbol{y}_j = \sum_i a_{i,j} x_i$; $\boldsymbol{y} = A^T X$ (shape: M×D)

Operations:

Alignments: $e_{i,j} = \boldsymbol{x}_i \cdot \boldsymbol{q}_j / \sqrt{D}$

$E = XQ^T/\sqrt{D}$ (shape: $N \times M$)

Attention: $a_{*,j} = \text{softmax}(e_{*,j})$

$A = \text{softmax}(E, dim = 1)$ $(N \times M)$

Inputs:

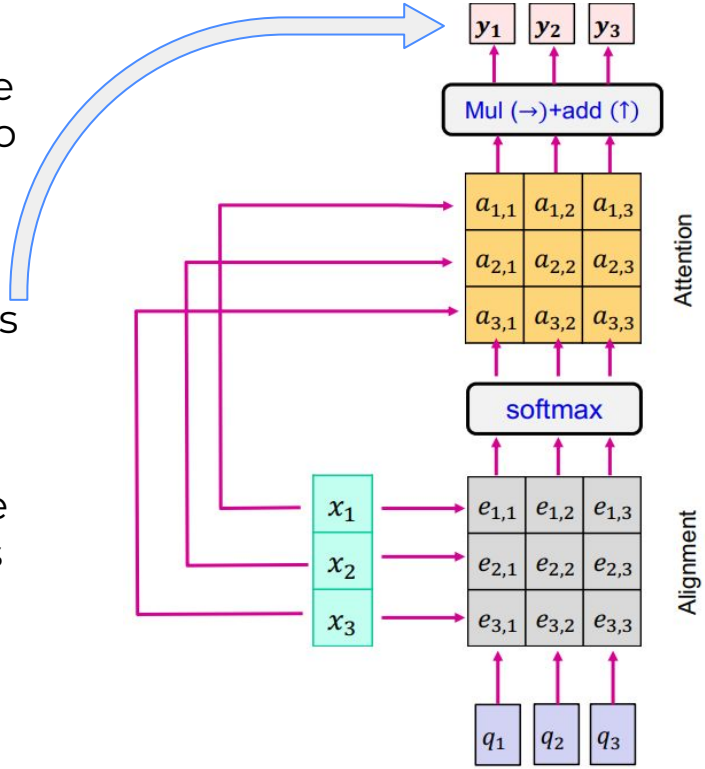
Input vectors: $\mathrm{X} = [x_1, ..]^T$ $(shape: N \times D)$

Query: $Q = [q_1, q_2, ...]^T$ $(shape: M \times D)$

Adapted from: Rienchman 3598

General Attention layer

- We can extend the attention layer into multiple query vectors
- Each query creates a new output context vector
- In this scheme the attention model is fixed and not learned



Outputs:

$y_j = \sum_i a_{i,j} x_i$; $y = A^T X$ (shape: M×D)

Operations:

Alignments: $e_{i,j} = x_i \cdot q_j / \sqrt{D}$

$E = XQ^T/\sqrt{D}$ (shape: $N \times M$)

Attention: $a_{*,j} = \text{softmax}(e_{*,j})$

$A = \text{softmax}(E, dim = 1)$ $(N \times M)$

Inputs:

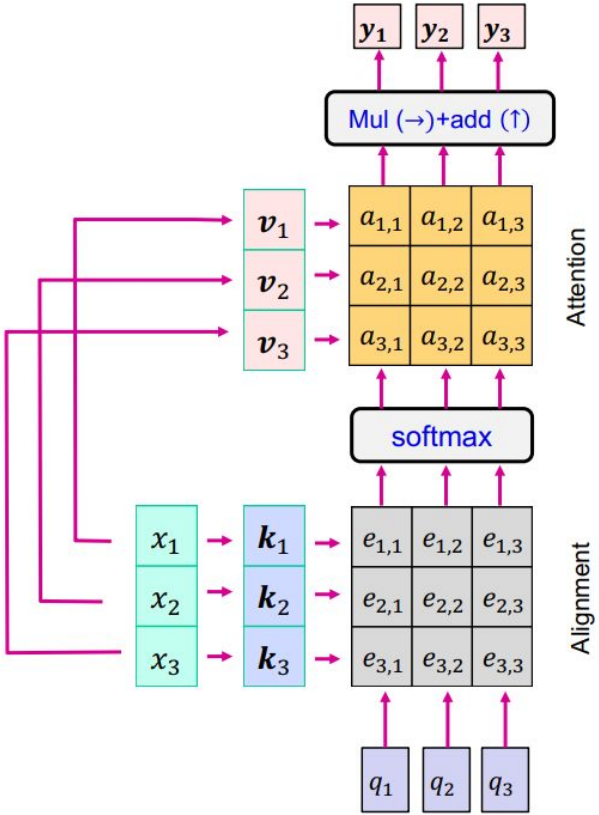
Input vectors: $X = [x_1, ..]^T$ $(shape: N \times D)$

Query: $Q = [q_1, q_2, ...]^T$ $(shape: M \times D)$

Adapted from: Rienchman 3598

General Attention layer

- We can add more expressivity by adding learnable weights to calculate the alignment and attention



Outputs:

$\boldsymbol{y}_j = \sum_i a_{i,j}\, \boldsymbol{v}_i$; Y (M×D_v)

Operations:

Key vectors: $\boldsymbol{k_i} = W_k\, \boldsymbol{x_i}$ (shape: D_k)

Value vectors: $\mathbf{v_i} = W_v\, \boldsymbol{x_i}$ (shape: D_v)

Alignments: $e_{i,j} = \boldsymbol{k}_i \cdot \boldsymbol{q}_j / \sqrt{D}$ ($N \times M$)

Attention: $a_{*,j} = \text{softmax}(e_{*,j})$ ($N \times M$)

Inputs:

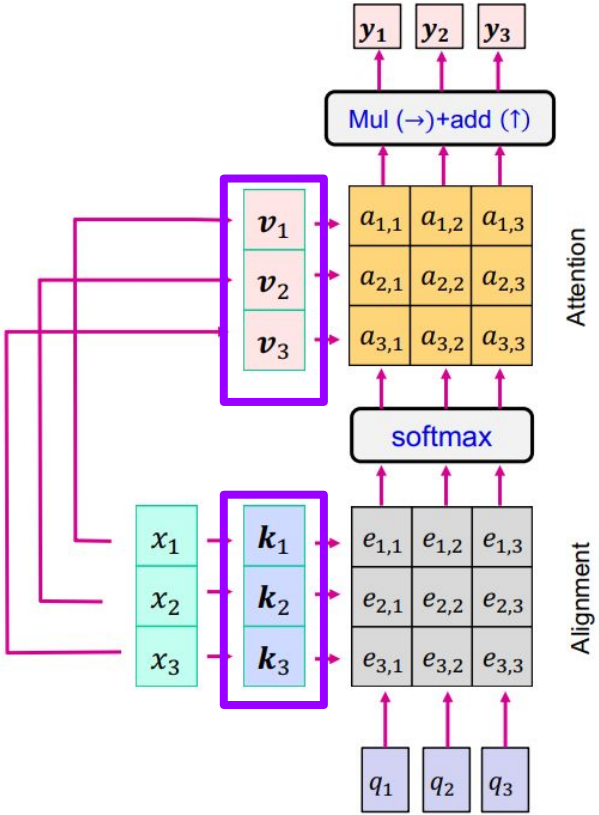
Input vectors: $X = [x_1, ..]^T$ ($shape: N \times D$)

Query: $\boldsymbol{Q} = [\boldsymbol{q_1}, \boldsymbol{q_2}, ...]^{\boldsymbol{T}}$ ($shape: M \times D_k$)

Adapted from: Rienchman 3598

General Attention layer

- We can add more expressivity by adding learnable weights to calculate the alignment and attention



Outputs:

$y_j = \sum_i a_{i,j} \, v_i$; Y (M×D_v)

Operations:

Key vectors: $k_i = W_k \, x_i$ (shape: D_k)

Value vectors: $v_i = W_v \, x_i$ (shape: D_v)

Alignments: $e_{i,j} = k_i \cdot q_j / \sqrt{D}$ ($N \times M$)

Attention: $a_{*,j} = \text{softmax}(e_{*,j})$ ($N \times M$)

Inputs:

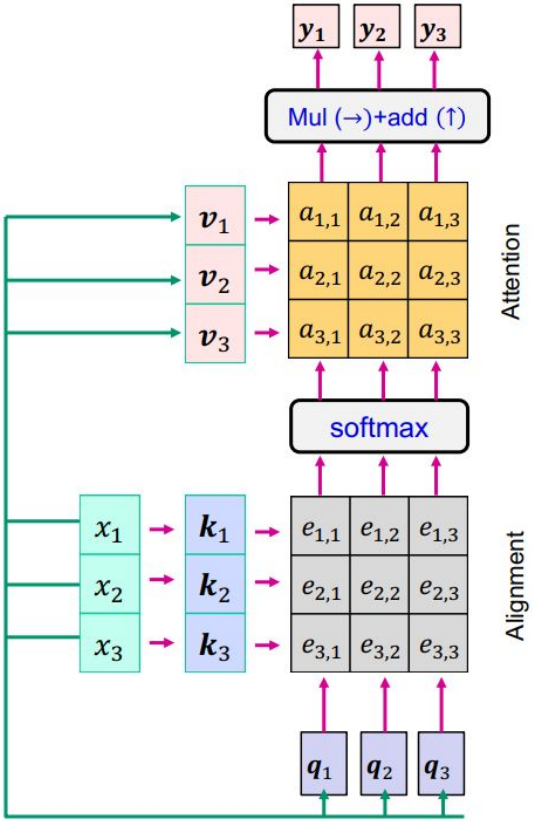
Input vectors: $X = [x_1, ..]^T$ ($shape: N \times D$)

Query: $Q = [q_1, q_2, ...]^T$ ($shape: M \times D_k$)

Adapted from: Rienchman 3598

Self Attention layer

- We calculate also the query vectors from the input vectors
- This defines a "self-attention" layer
- Query vectors: $\boldsymbol{q} = W_q\, \boldsymbol{x}$
- Is used to capture context within the sequence: [x_1, x_2, x_3]



Outputs:

Multiple Context vectors: $\boldsymbol{y}$ $(N \times D)$

Operations:

Query vectors: $\boldsymbol{q} = W_q\, \boldsymbol{x}$ $(N \times D_k)$

Key vectors: $\boldsymbol{k} = W_k\, \boldsymbol{x}$ $(N \times D_k)$

Value vectors: $\mathbf{v} = W_v\, \boldsymbol{x}$ $(N \times D)$

Alignments: $e_{i,j} = \boldsymbol{k}_i \cdot \boldsymbol{q}_j / \sqrt{D}$

Attention: $a_{*,j} = \mathrm{softmax}(e_{*,j})$

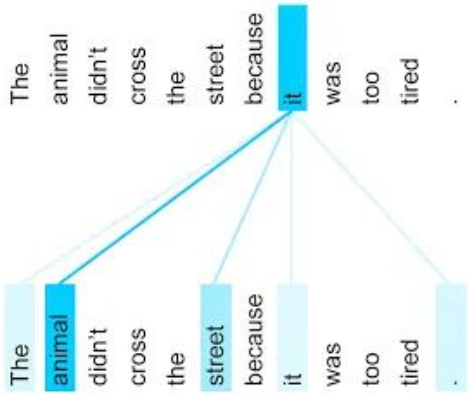
Output: $\boldsymbol{y}_j = \sum_i a_{i,j}\, \boldsymbol{v}_i$

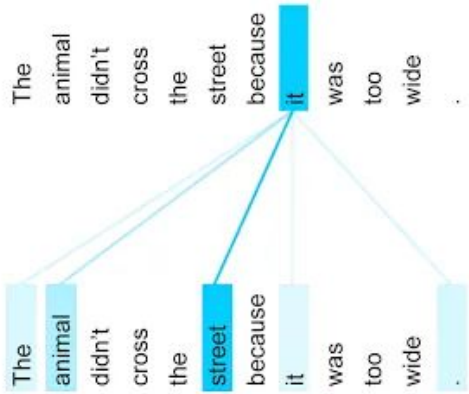
Inputs:

Input vectors: $\boldsymbol{x}$ $(shape: N \times D)$

Adapted from: Rienchman 3598

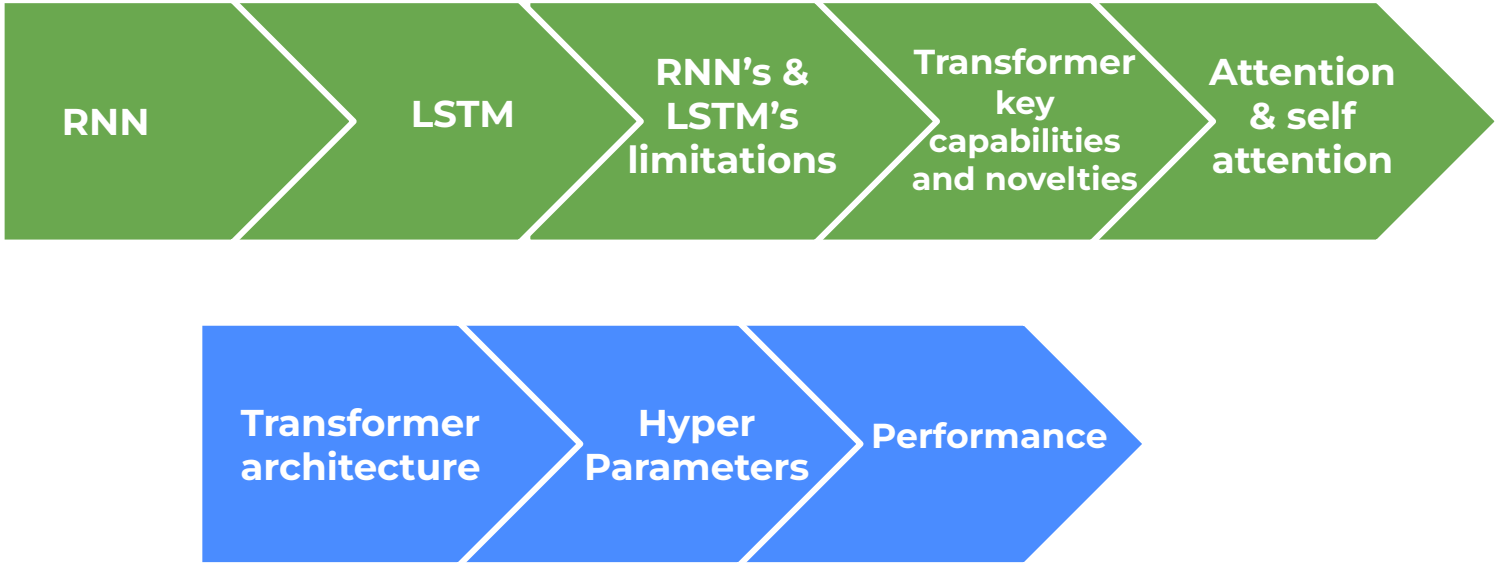
Self Attention layer





Adapted from: Rienchman 3598

In This Chapter



Transformer Architecture - Attention is all you need

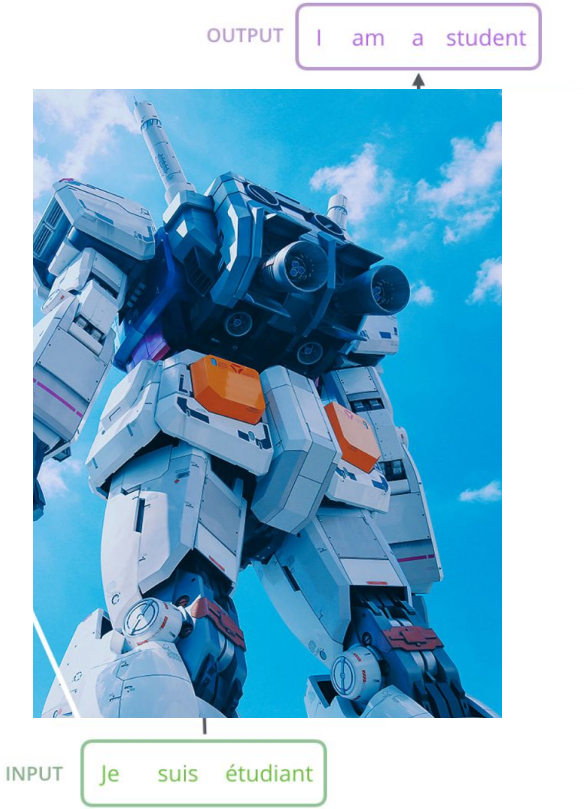


Image Source: FPPT.com

Transformer Architecture

An Encoder Decoder model, usually stacked

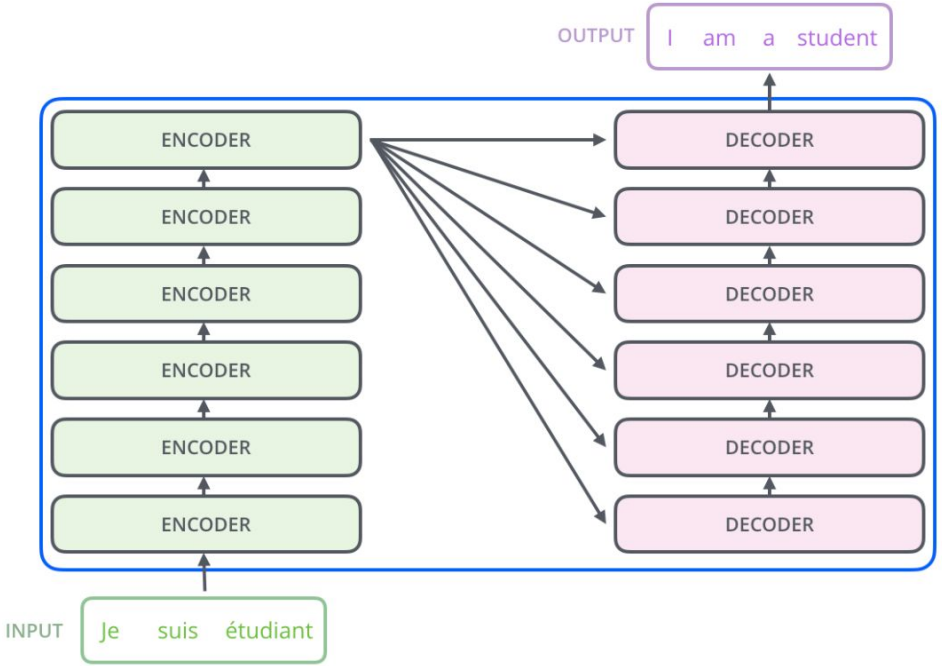


Image Source: Jay alammar

Transformer Architecture

Each Encoder Decoder consists of:

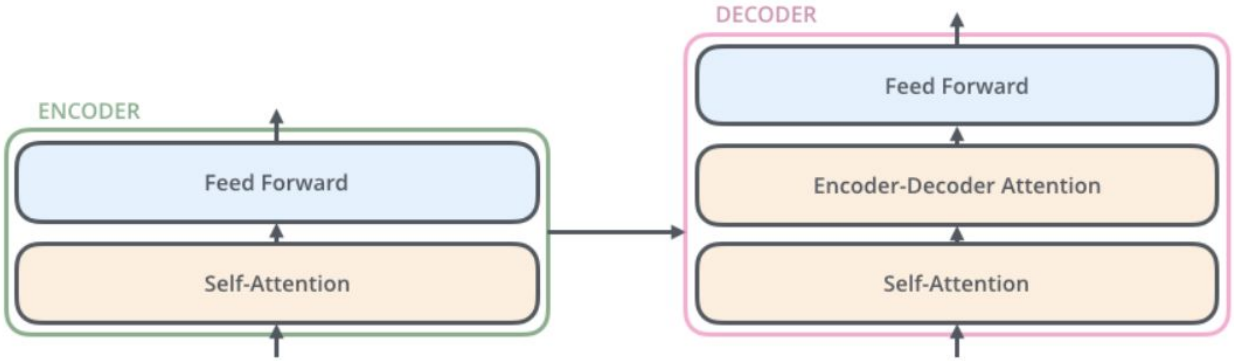
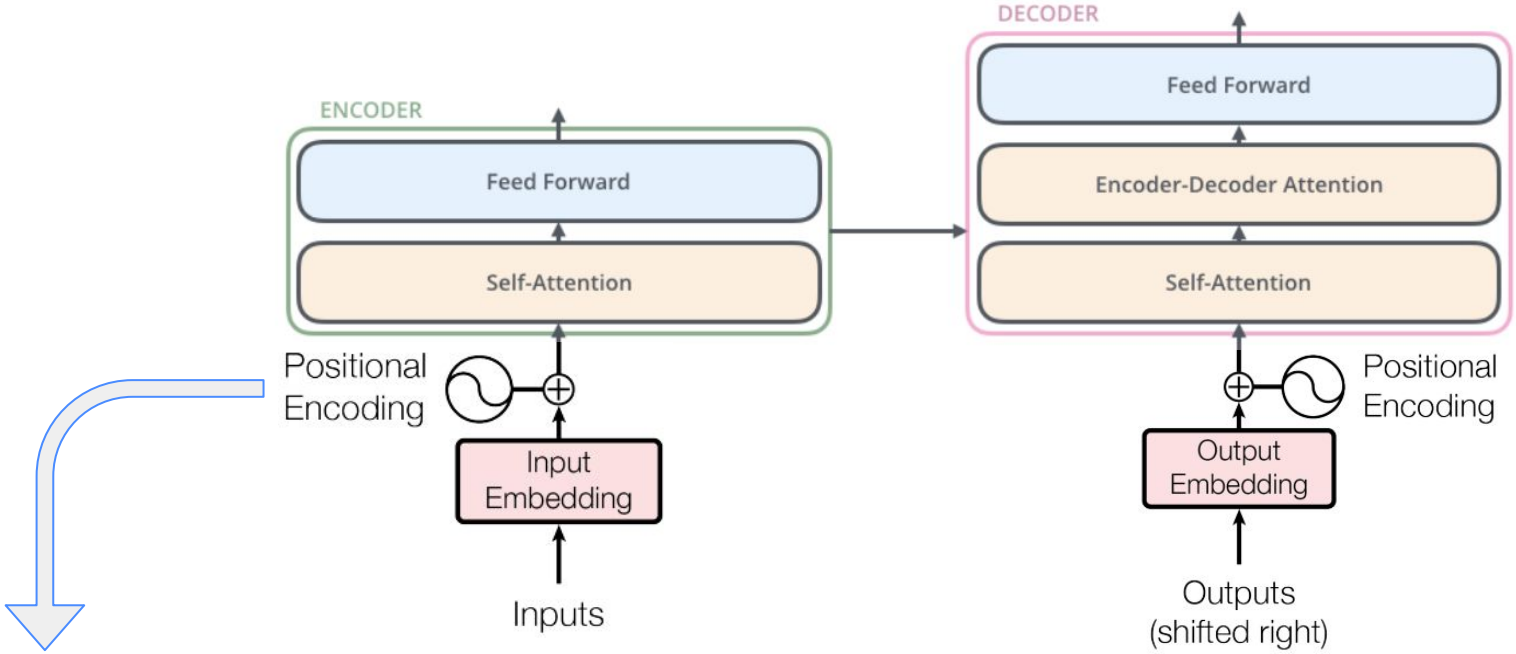


Image Source: Jay alammar

Transformer Architecture



No recurrence or convolution is used - in order for the model to make use of the order of the sequence, relative or absolute position must be injected

Image Source: Jay alammar

Positional encoding

- Requirements
  - Every position has a unique position encoding vector
  - Relative distances between positions are maintained
- Possible Solutions: Indexing, Fraction of sentence length, frequency base

Can overtone semantic embedding!

Dependant on input length!

Chosen solution

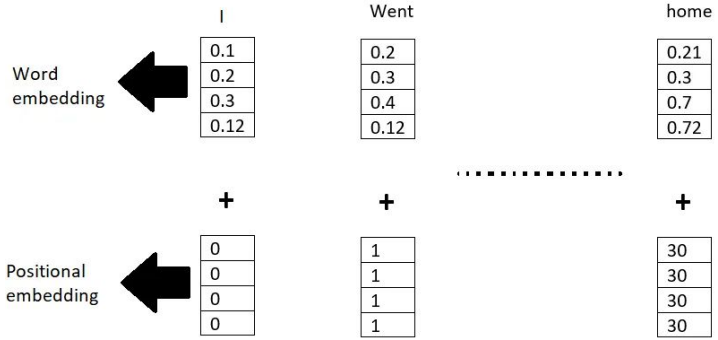


Image Source: Towards Science

Positional encoding

- The Sinusoidal function
- For each position we compute a vector of dimension d_{model} s.t :

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

Where

d_{model} = size of the embedding
pos = token position
i = is the embedding index dim

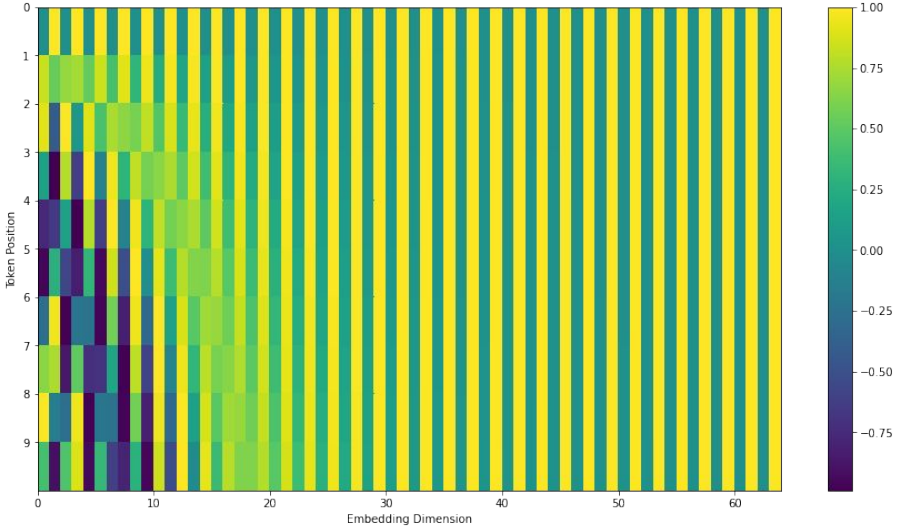
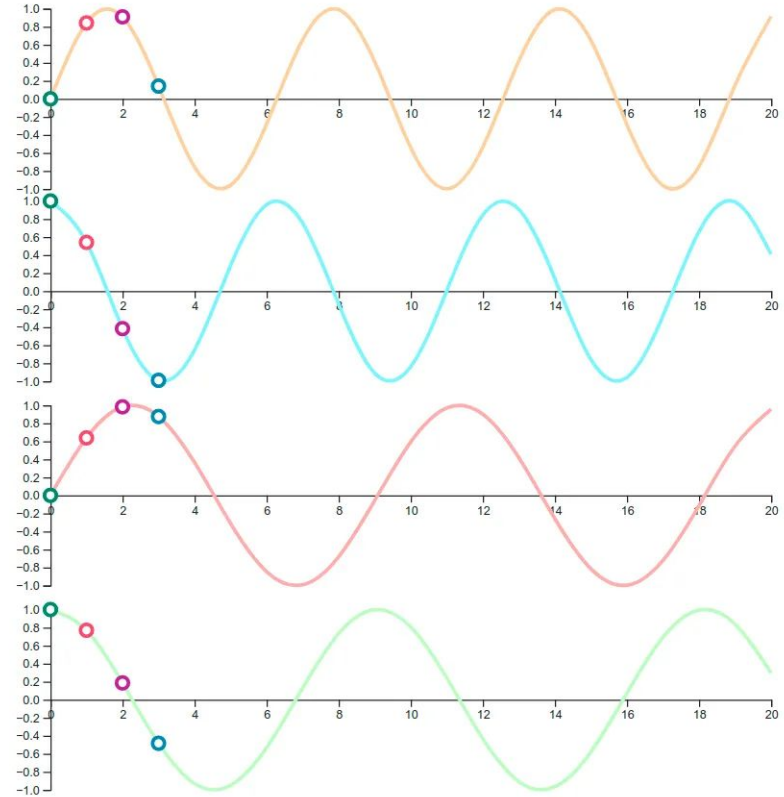


Image Source: Jay Alammar

Positional encoding



	p0	p1	p2	p3
i=0	0.000	0.841	0.909	0.141
i=1	1.000	0.540	-0.416	-0.990
i=2	0.000	0.638	0.983	0.875
i=3	1.000	0.770	0.186	-0.484

Positional Encoding

$$PE_{(pos,2i)} = sin(\frac{pos}{10000^{2i/d_{\text{model}}}})$$

$$PE_{(pos,2i+1)} = cos(\frac{pos}{10000^{2i/d_{\text{model}}}})$$

Settings: d = 50

The value of each positional encoding depends on the *position* (*pos*) and *dimension* (*d*). We calculate result for every *index* (*i*) to get the whole vector.

Image Source: Towards Science

Positional encoding

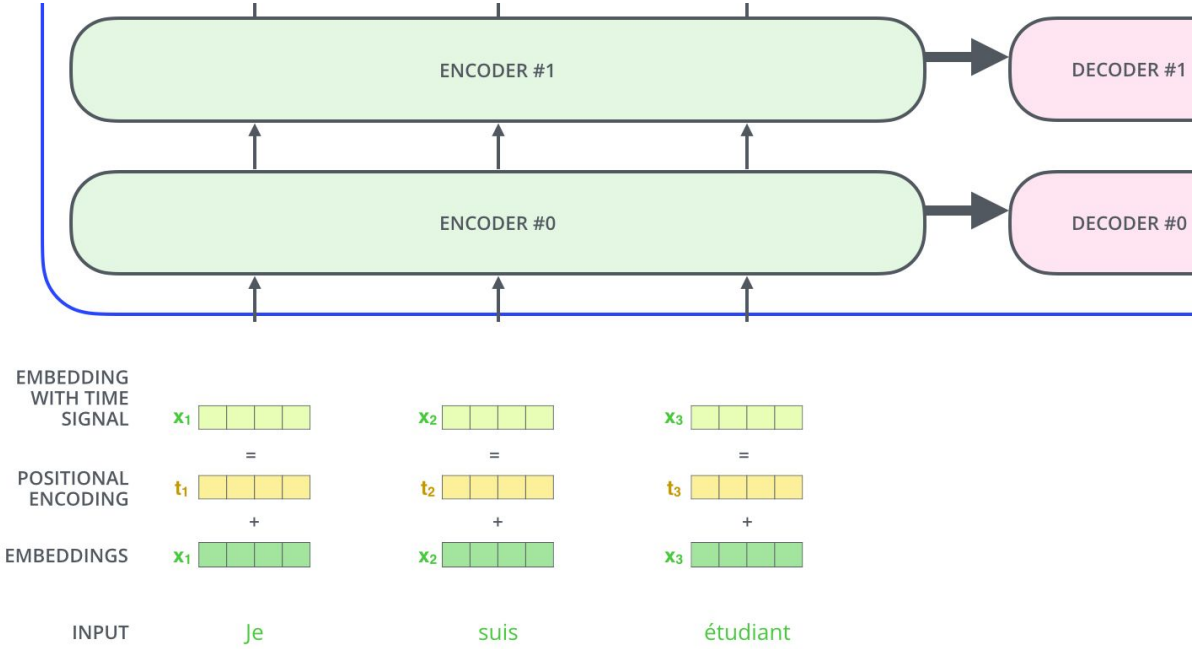
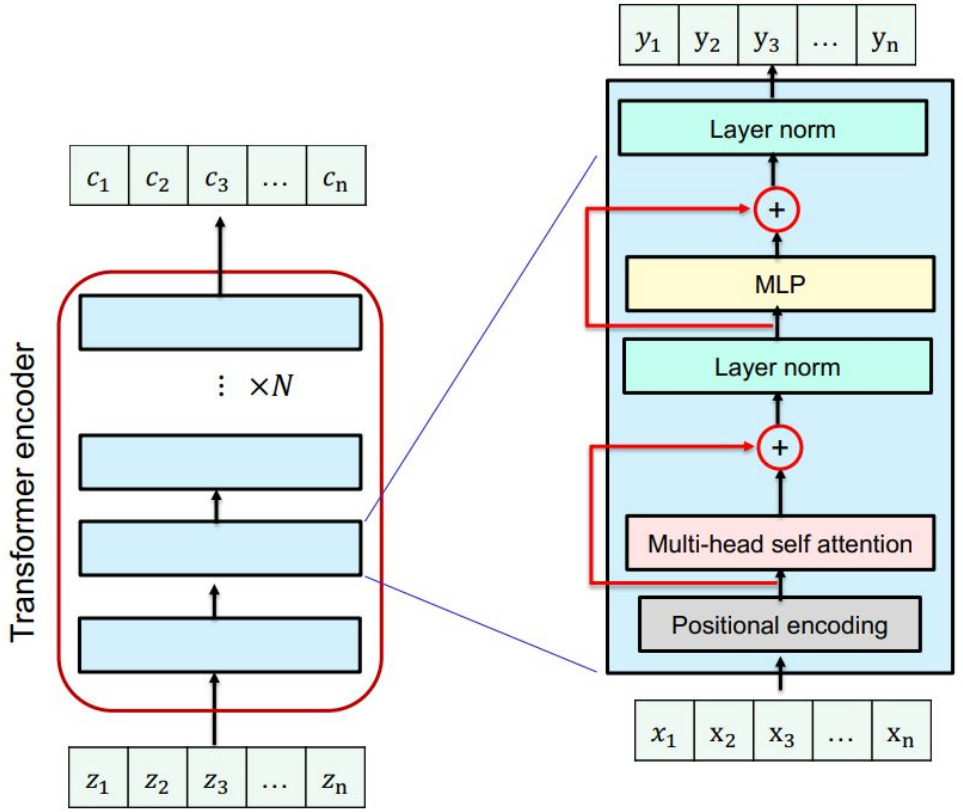


Image Source: Jay Alammar

Transformer Architecture - Encoder block

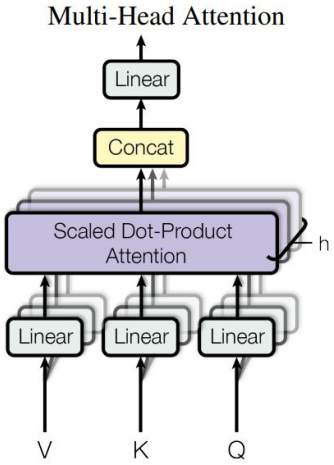
- Multi-head Self attention
- the Attention module repeats its computations multiple times in parallel, each of these is called an Attention Head
- In the paper they used $d_k = d_v = d_{model}/h = 64$
- Due to the reduced dimension of each head, the total computational cost is similar to that of single-head attention with full dimensionality
- Multiple smaller heads seems to work better!

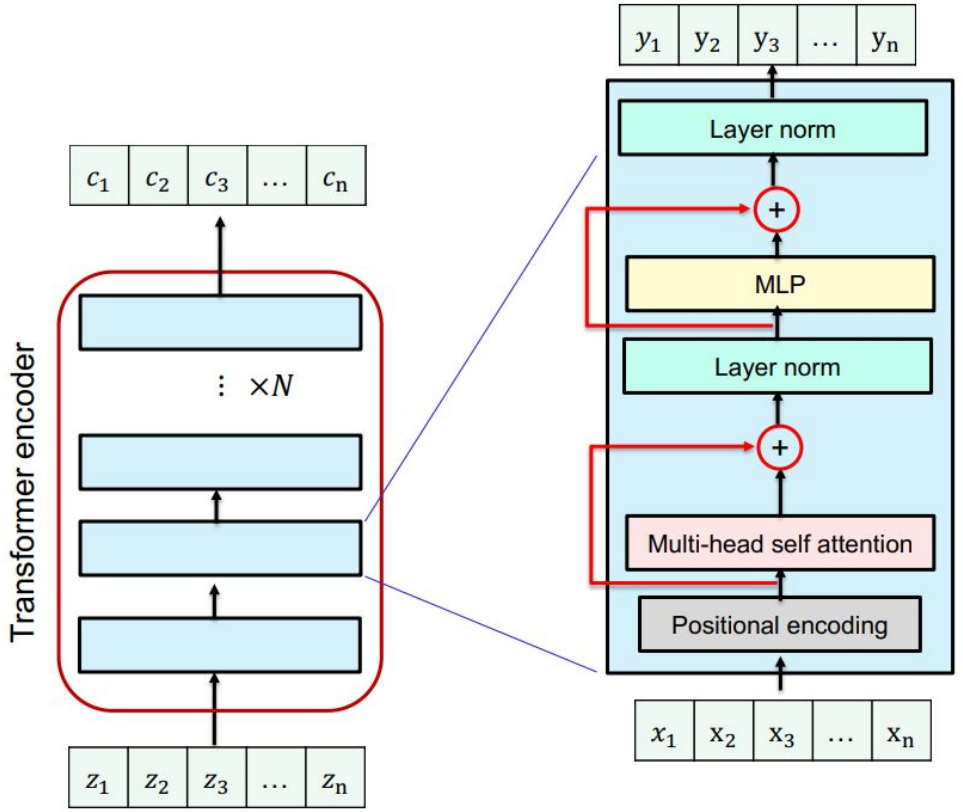


Adapted from: Rienchman 3598

Transformer Architecture - Encoder block

- Multi-head Self attention
- the Attention module repeats its computations multiple times in parallel, each of these is called an Attention Head





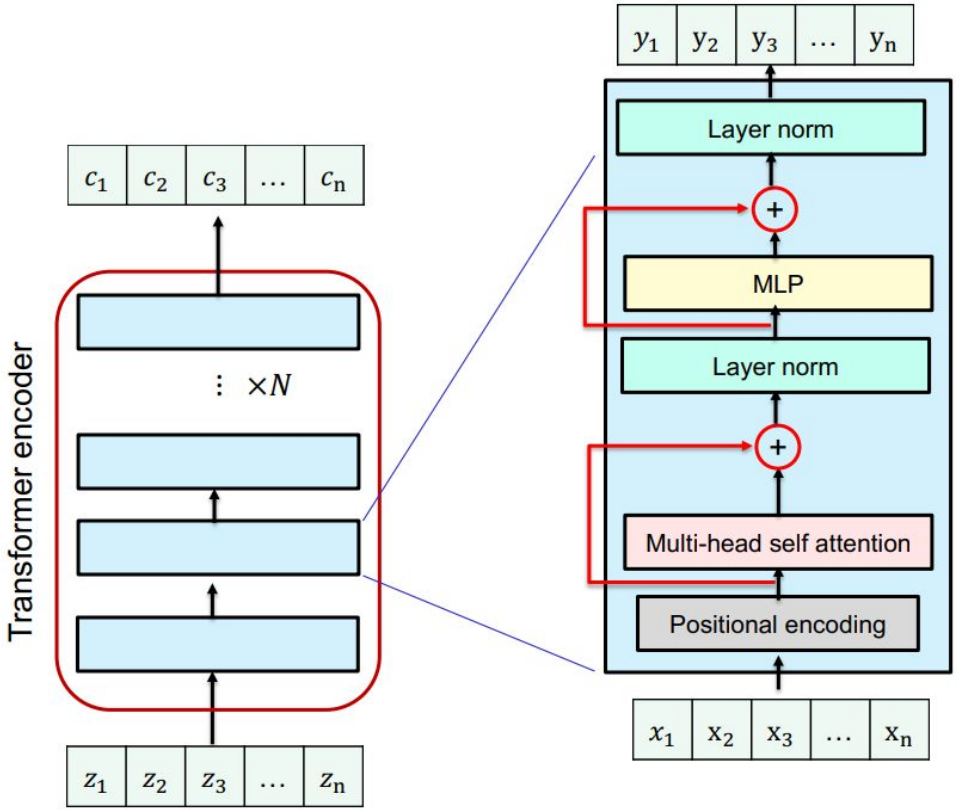
Adapted from: Rienchman 3598

Transformer Architecture - Encoder block

- Multi-head Self attention

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

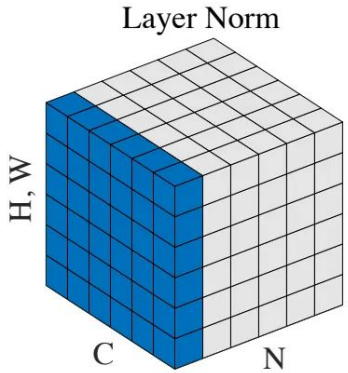
$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

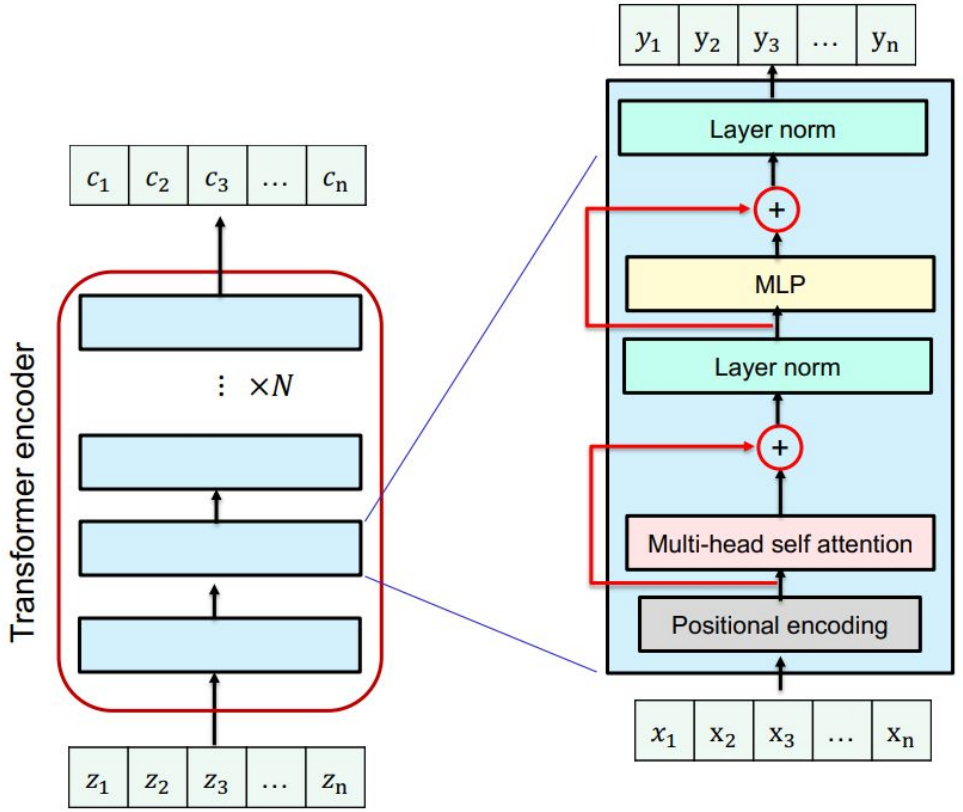


Adapted from: Rienchman 3598

Transformer Architecture - Encoder block

- Layer Norm
- Invariant to batch size and sequence length
- Input values in all neurons in the same layer are normalized for each data sample

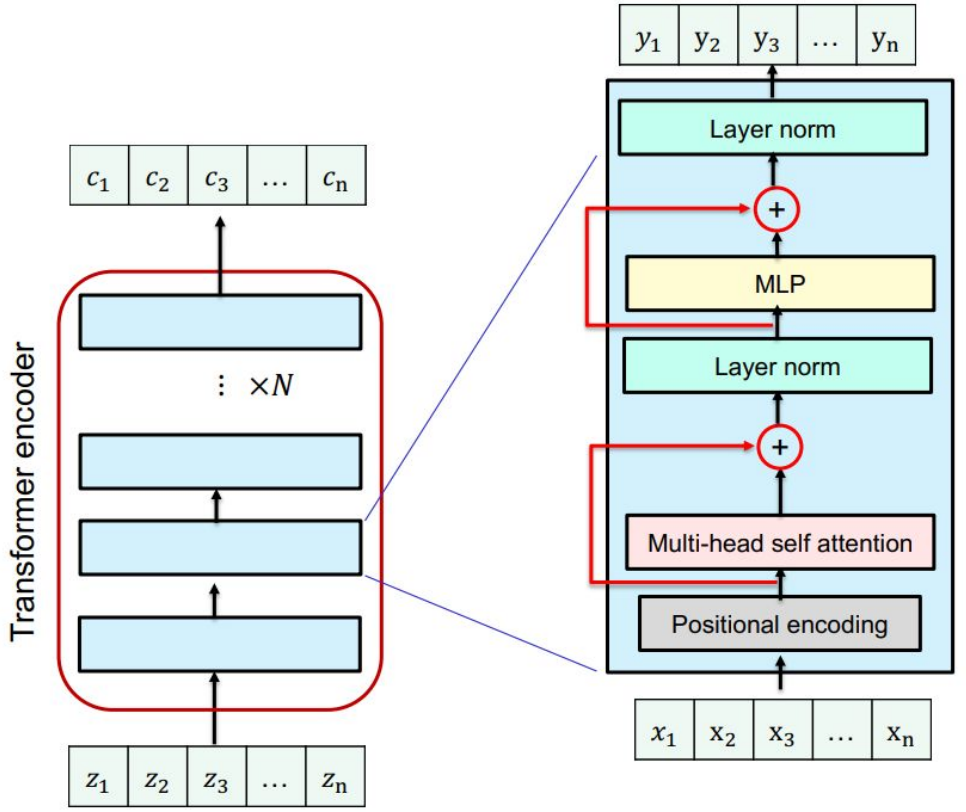




Adapted from: Rienchman 3598

Transformer Architecture - Encoder block

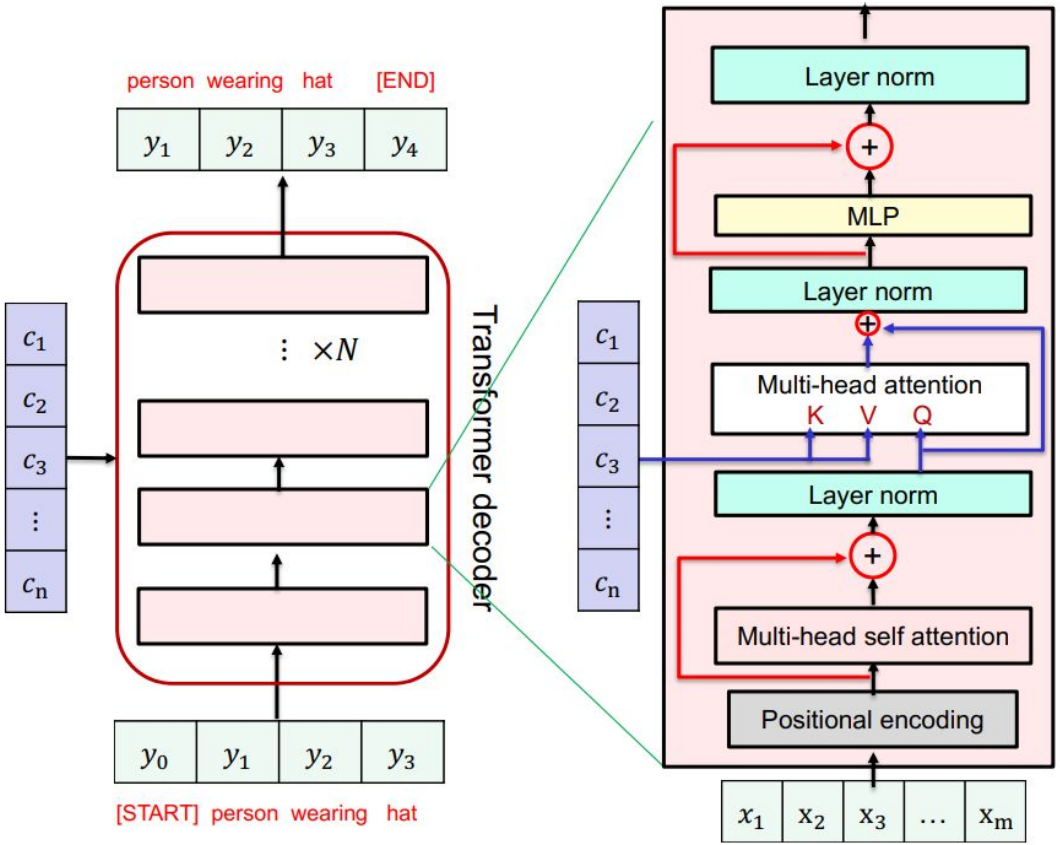
- Multi-head Self attention
- N = 6 blocks
  embedding dimension = 512
  8 attention heads



Adapted from: Rienchman 3598

Transformer Architecture - Decoder block

- Multi-head self attention modified to mask subsequent positions (prevent reverse information flow)
- Multi-head attention block is NOT self attention, it attends over encoder outputs
- N = 6 blocks
  embedding dimension = 512
  8 attention heads



Adapted from: Rienchman 3598

Transformer Architecture

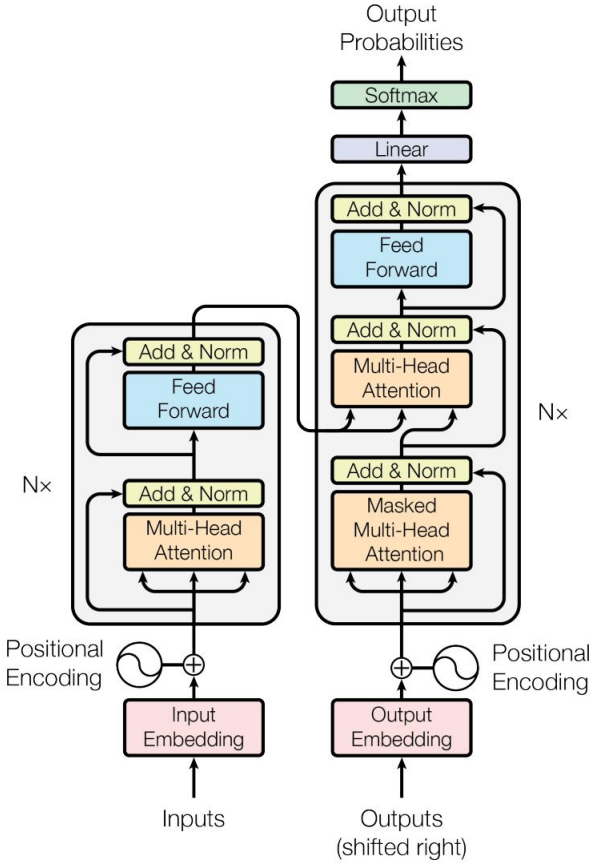


Image Source: Attention is all you need

Transformer Architecture - Decoder output

- The output is forwarded to a fully connected neural network that projects the vector produced by the stack of decoders into a much larger vector called a logits vector
- This vector holds the scores of all the unique words in the output vocabulary
- Let’s assume that our model knows 10,000 unique English words, his would make the logits vector 10,000 cells wide
- From there it is passed to a Softmax turning those scores into probabilities
- The cell with the highest probability is chosen

Transformer Architecture - Decoder output

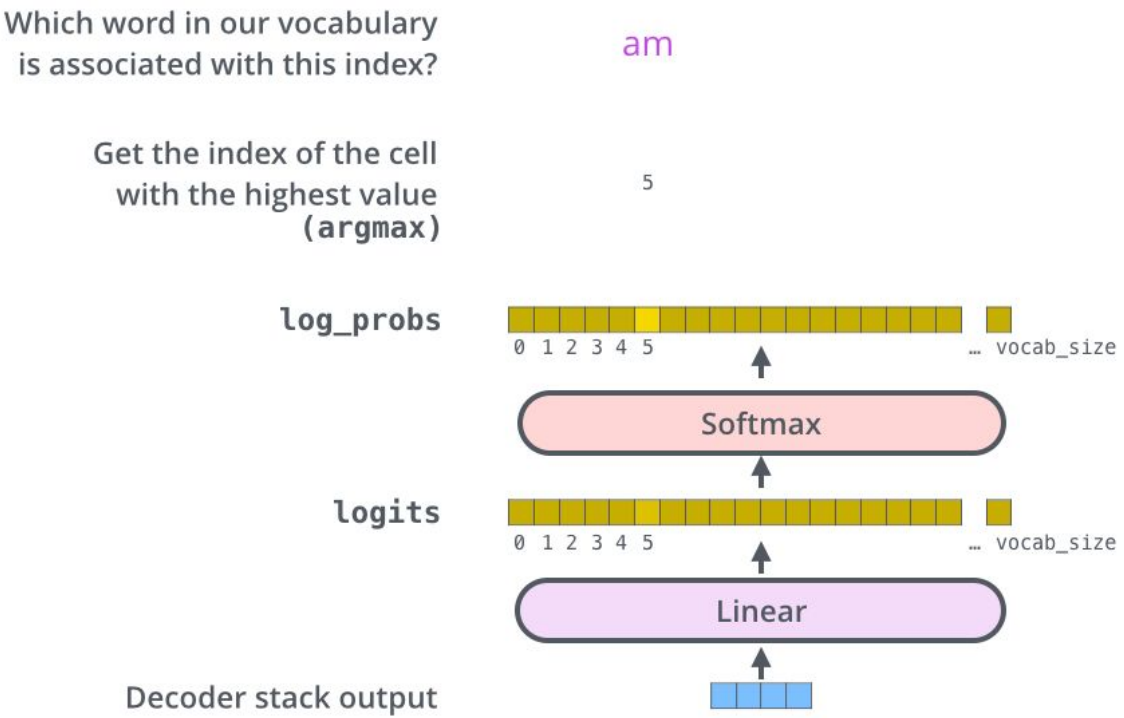
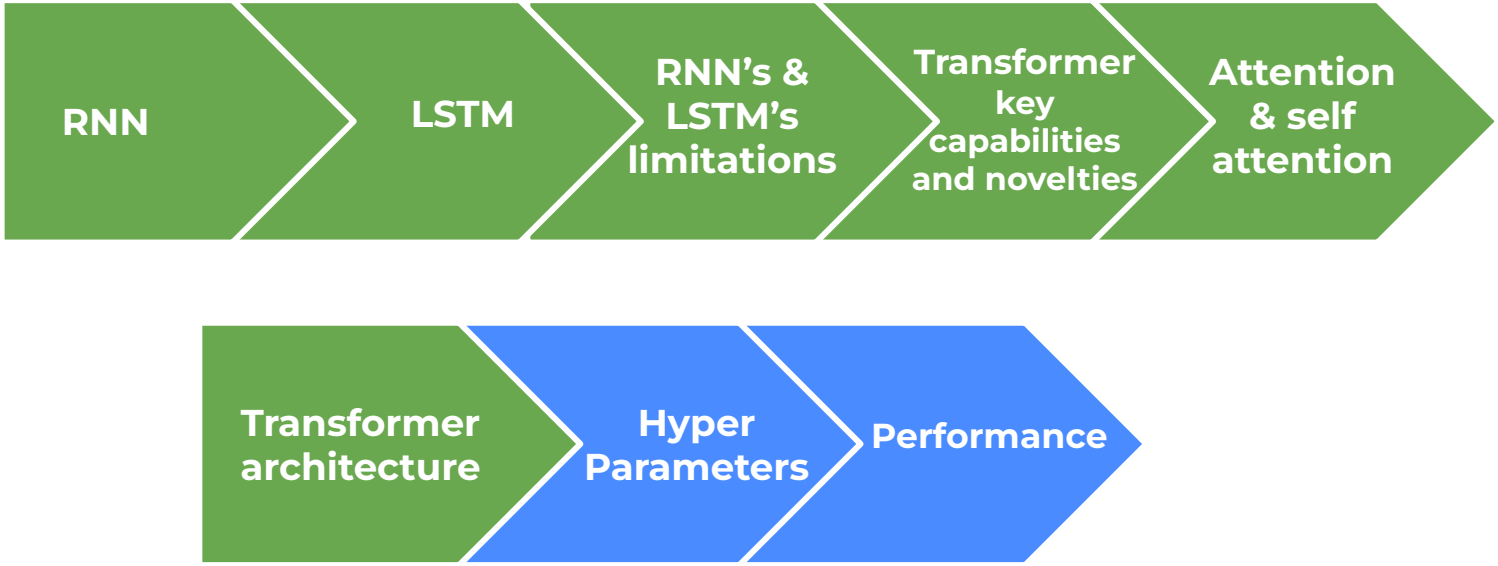


Image Source: Jay Alammar

In This Chapter



Attention is all you need - Hyper params

- WMT 2014 English-to-German translation task - 4.5 million sentence pairs
- WMT 2014 English-to-French translation task - 36M sentence pairs
- **Batching**

  25000 source & target tokens with dynamic batching to minimize padding

- **Optimizer**

  Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.98$, $\varepsilon = 10^{-9}$ hyperparameters

- **Regularization**

  Residual Dropout p=0.1, Label Smoothing $\varepsilon_{ls} = 0.1$

Attention is all you need - Hyper params

- WMT 2014 English-to-German translation task - 4.5 million sentence pairs
- WMT 2014 English-to-French translation task - 36M sentence pairs
- **Batching**

  25000 source & target tokens with dynamic batching to minimize padding

- **Optimizer**

  Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.98$, $\varepsilon = 10^{-9}$ hyperparameters

- **Regularization**

  Residual Dropout p=0.1, Label Smoothing $\varepsilon_{ls} = 0.1$

  ↓

  Introduces noise to Softmax with **K** output values
  by replacing the hard 0 and 1 classification targets
  with targets of $\boldsymbol{\varepsilon/K-1}$ and $\boldsymbol{1-\varepsilon}$ respectively

Attention is all you need - Model Performance

- **BLEU score of 28.4** on English-German translation task, outperforming the previous best model by a significant margin
- BLEU = BiLingual Evaluation Understudy
- A number that measures the similarity of the machine-translated text to a high quality reference translation based on n-gram precision (n=4)

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [15]	23.75			
Deep-Att + PosUnk [32]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [31]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [8]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [26]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [32]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [31]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [8]	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**	**41.0**	$2.3 \cdot 10^{19}$	

$$Bleu\ (N) = Brevity\ Penalty \cdot Geometric\ Average\ Precision\ Scores\ (N)$$

Image Source: Attention is all you need

Attention is all you need - Model Performance

- **BLEU score of 28.4** on English-German translation task, outperforming the previous best model by a significant margin

$$Bleu\ (N) = Brevity\ Penalty \cdot Geometric\ Average\ Precision\ Scores\ (N)$$

BLEU Score	Interpretation
30 - 40	Understandable to good translations
40 - 50	High quality translations
50 - 60	Very high quality, adequate, and fluent translations
> 60	Quality often better than human

Attention is all you need - Model Performance

- **BLEU score of 28.4** on English-German translation task, outperforming the previous best model by a significant margin
- **Significantly faster to train**

  3.5 days using 8 NVIDIA P100 GPUs VS **1 week** using 32 NVIDIA K40 GPUs
  P100 ≈ 1.5 × K40
- **Training cost FLOPS**

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [15]	23.75			
Deep-Att + PosUnk [32]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [31]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [8]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [26]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [32]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [31]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [8]	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**	**41.0**	$2.3 \cdot 10^{19}$	

Image Source: Attention is all you need

Questions?

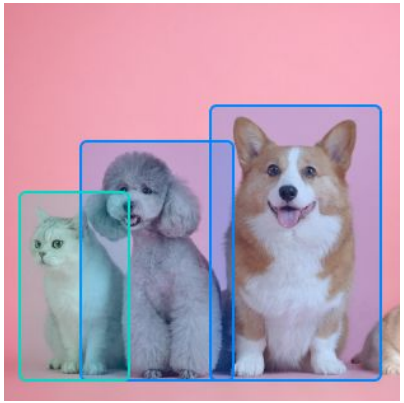
02
Vision Transformer

In This Chapter

CV Main Tasks

Classification

Detection

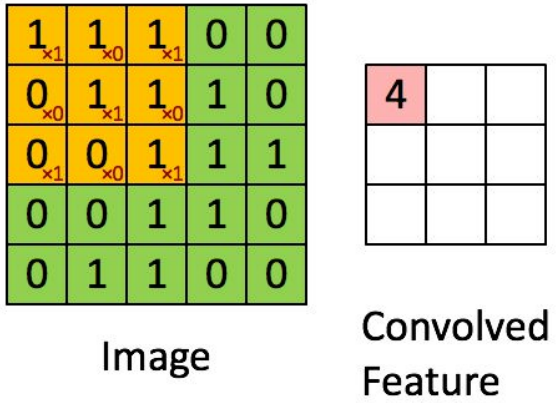


Segmentation

In This Chapter

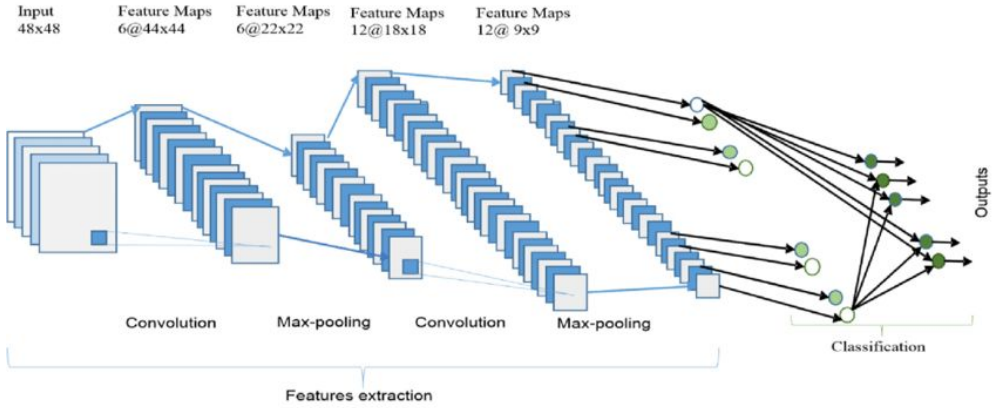
CNN - Convolution Layer

- A k x k Kernel (usually small) that shifts through the image
- In each step - multiply corresponding cells with kernel weights and then sum
  - Stride = the step size
  - Padding = affects the output size
- Input size = n x m
- Output size with stride 1 = (n-k+1) X (m-k+1)

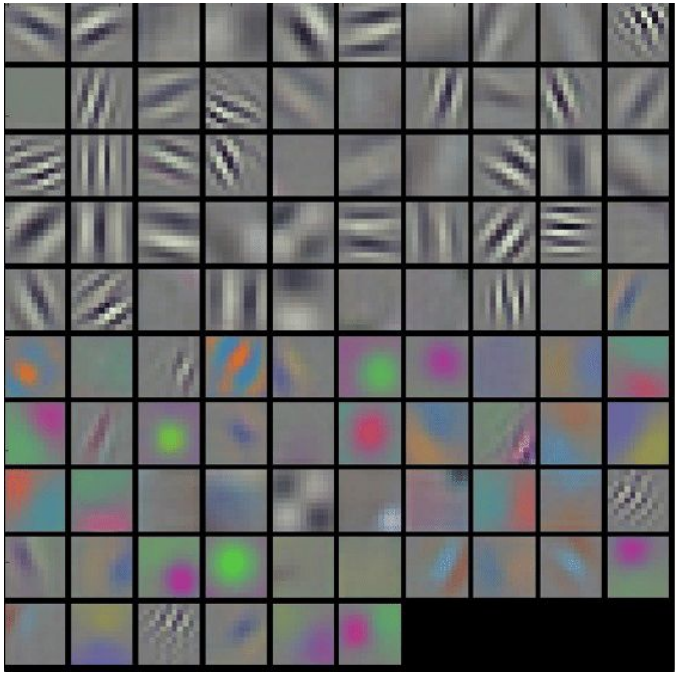


CNN - Objective

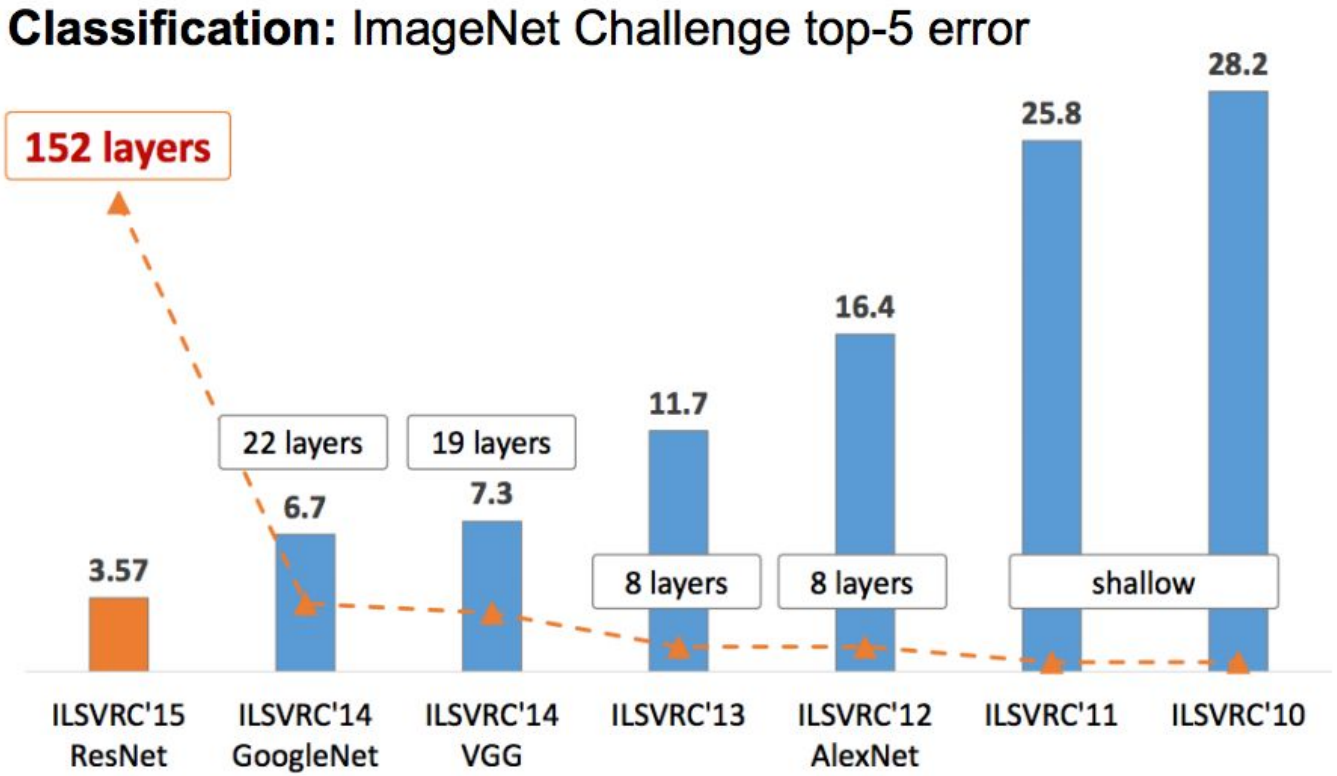
- We want to extract some features (e.g., color histogram, edges) first, and use them as an input to our classifier
- The first ConvLayer is responsible for capturing the spatial features such as edges, color, gradient orientation, etc
- With added layers, the architecture adapts to the semantic features as well, giving us a network that has a wholesome understanding of images in the dataset, similar to how humans would



AlexNet first layer filters



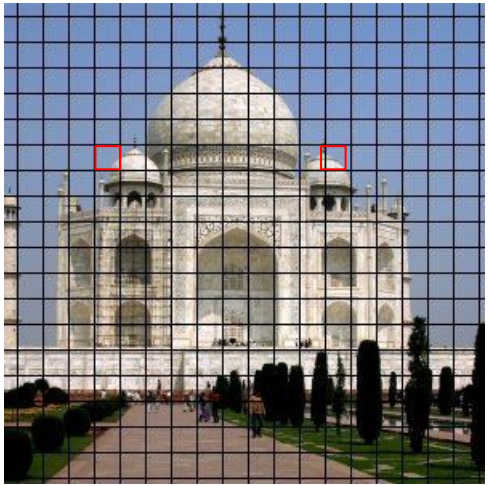
Deeper is better?



In This Chapter

Vision Transformer - Motivation

- Transformer architecture has become the de-facto standard for natural language processing tasks, why not to use it for images?
- CNN only attend to “neighbours” pixels in each layer
- Transformers can relate to “far” pixels in single layer



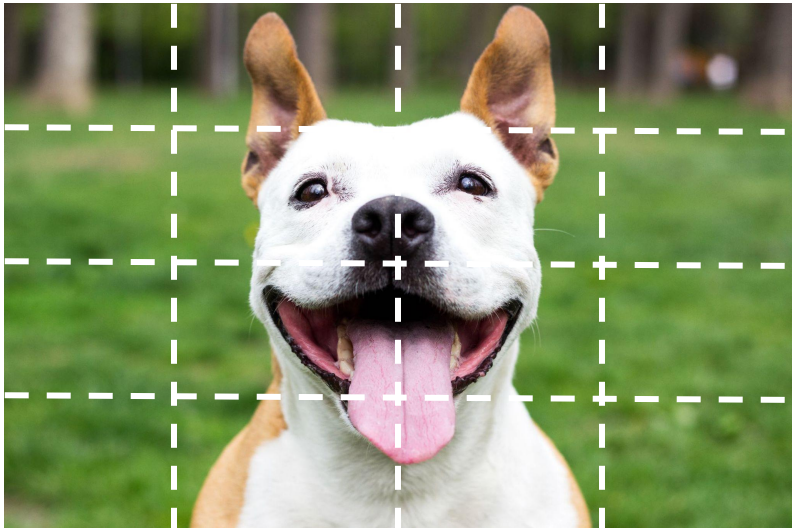
Vision Transformer - Naive Idea

- Feeding the Transformers with all image pixels and calculate the attention (relation) between different pixels
- Main problem - for 250X250 image the attention metrics calculating cost is

  $(250^2)^2 = 3{,}906{,}250{,}000$

Vision Transformer - Main Idea

- Splitting the image into patches
- Calculate attention between patches with “standard” Transformer
  - Patch = token/word
  - Image = sentence
- Predict class with the Transformer output



In This Chapter

Vision Transformer Architecture

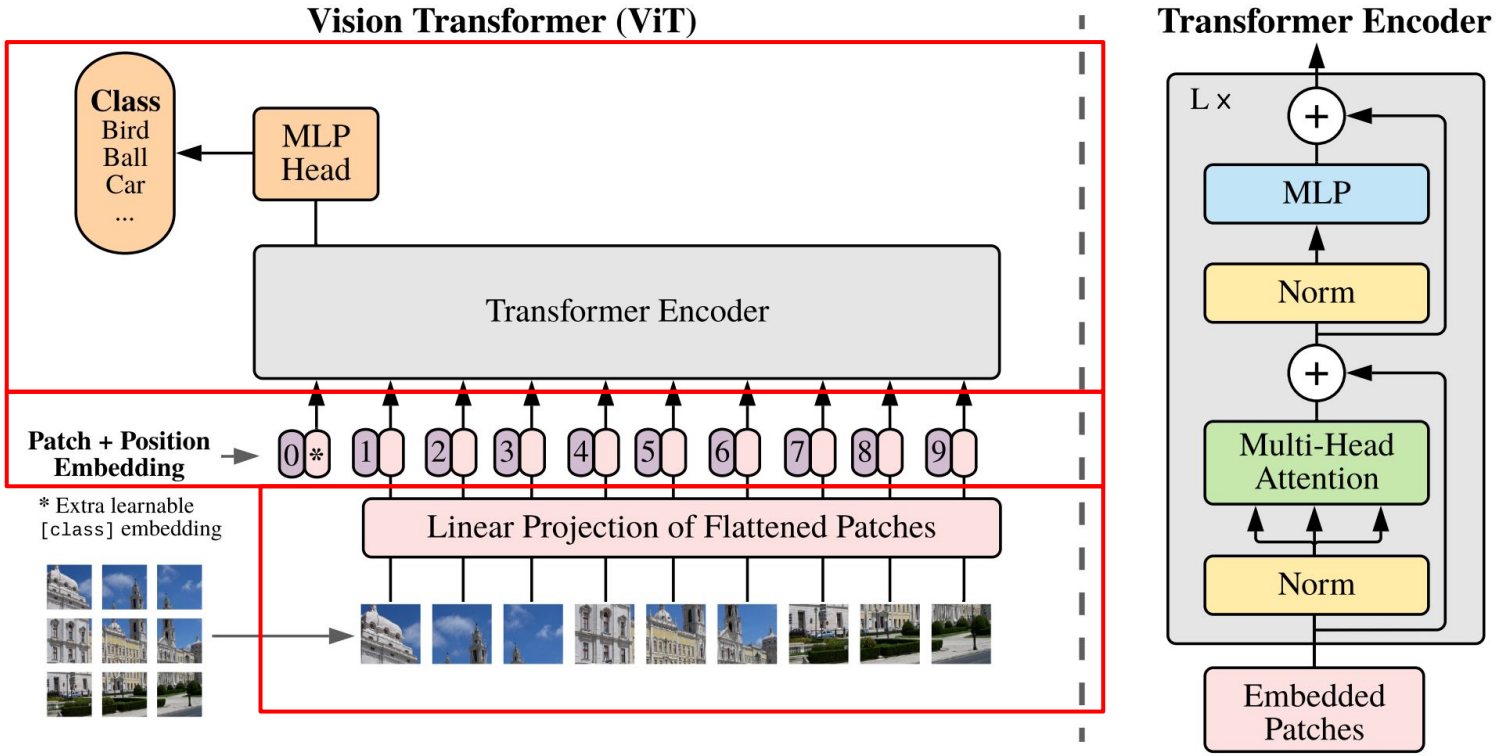
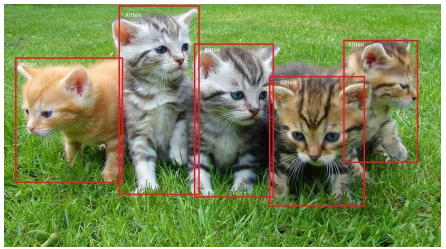


Image Source: An Image is Worth 16X16 words

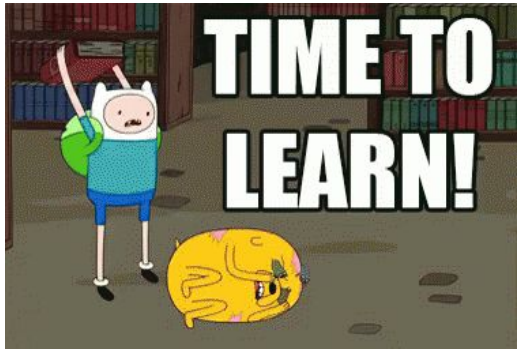
Inductive Bias

- The inductive bias of a learning algorithm is the set of assumptions that the learner uses to predict outputs of given inputs that it has not encountered
- In other words - prior knowledge or assumptions that are built into the learning algorithm, which guide the learning process and shape the resulting model
    - For example, a linear regression model assumes that the relationship between the input variables and the output variable is linear. This is a type of inductive bias
- In CNN - Translation Equivariance is an inductive bias that allows the model to recognize patterns or features in an image regardless of their position within the image.



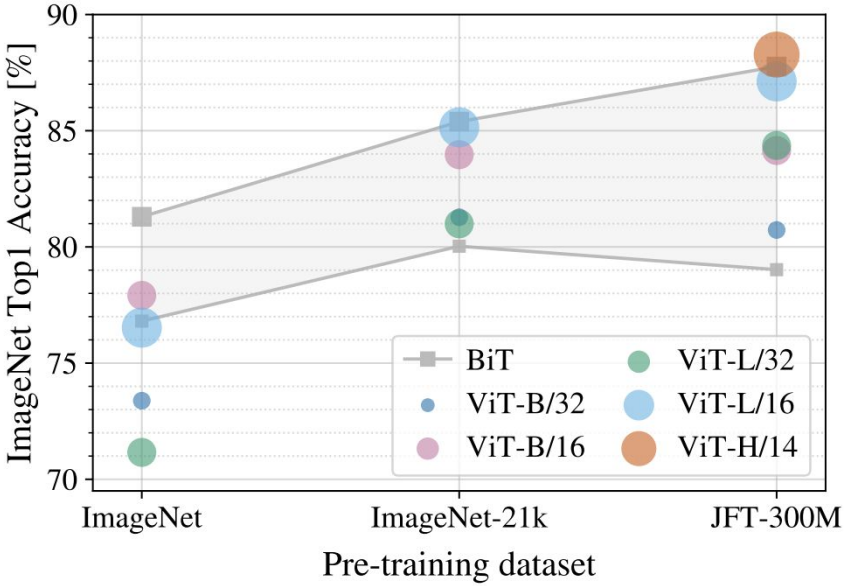
Training the model - Transfer Learning

- ViT lack some of the inductive biases inherent to CNNs, therefore when trained on mid-sized datasets such as ImageNet, ViT yields modest accuracies of a few percentage points below ResNets of comparable size
- However, the picture changes if the models are trained on larger datasets (14M-300M images)
- Therefore, ViT are pre-trained at sufficient scale data-sets and then fine-tuned to smaller tasks (Transfer Learning)



In This Chapter

Size Matters



Small DS → **Big DS**

Image Source: An Image is Worth 16X16 words

More accuracy, less training time

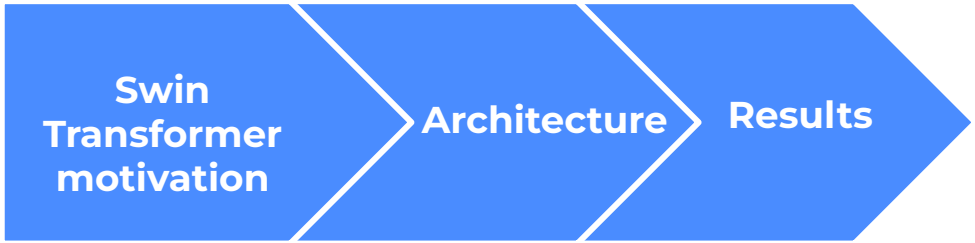
	Ours-JFT (ViT-H/14)	Ours-JFT (ViT-L/16)	Ours-I21k (ViT-L/16)	BiT-L (ResNet152x4)	Noisy Student (EfficientNet-L2)
ImageNet	**88.55** ± 0.04	87.76 ± 0.03	85.30 ± 0.02	87.54 ± 0.02	88.4/88.5*
ImageNet ReaL	**90.72** ± 0.05	90.54 ± 0.03	88.62 ± 0.05	90.54	90.55
CIFAR-10	**99.50** ± 0.06	99.42 ± 0.03	99.15 ± 0.03	99.37 ± 0.06	–
CIFAR-100	**94.55** ± 0.04	93.90 ± 0.05	93.25 ± 0.05	93.51 ± 0.08	–
Oxford-IIIT Pets	**97.56** ± 0.03	97.32 ± 0.11	94.67 ± 0.15	96.62 ± 0.23	–
Oxford Flowers-102	99.68 ± 0.02	**99.74** ± 0.00	99.61 ± 0.02	99.63 ± 0.03	–
VTAB (19 tasks)	**77.63** ± 0.23	76.28 ± 0.46	72.72 ± 0.21	76.29 ± 1.70	–
TPUv3-core-days	2.5k	0.68k	0.23k	9.9k	12.3k

Image Source: An Image is Worth 16X16 words

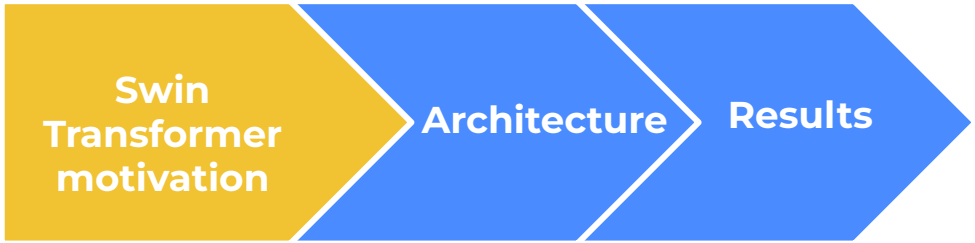
Questions?

03
Swin Transformer

In This Chapter

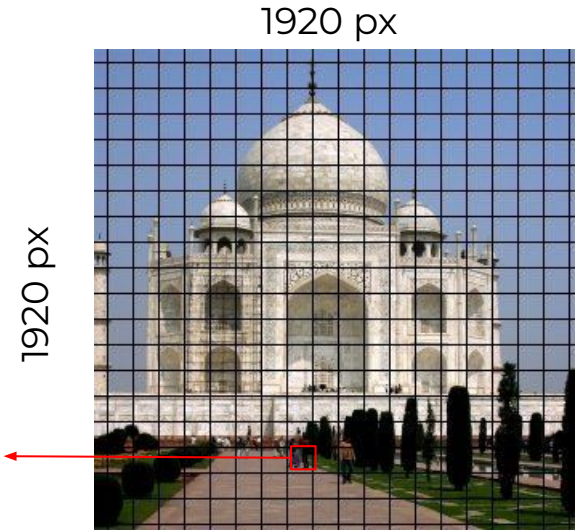


In This Chapter



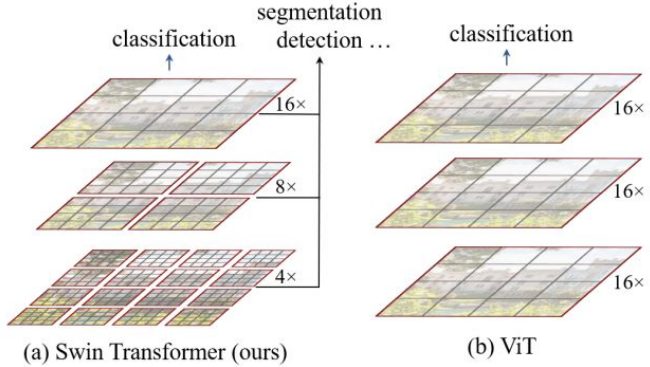
Swin Transformer - Motivation

- Main problems in ViT:
  - High resolution images = large computational time
  - Recognizing small elements in the image



Swin Transformer - Proposed Solutions

- Small Element Recognition
  - Instead of fixed patch size, we use varying size patches. starting from small-sized patches and merging them on deeper layers



- Computational Time
  - Instead of calculating global patches attention, we use local attention with shifted “local” windows through layers

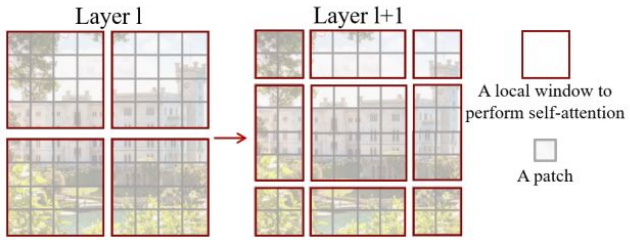
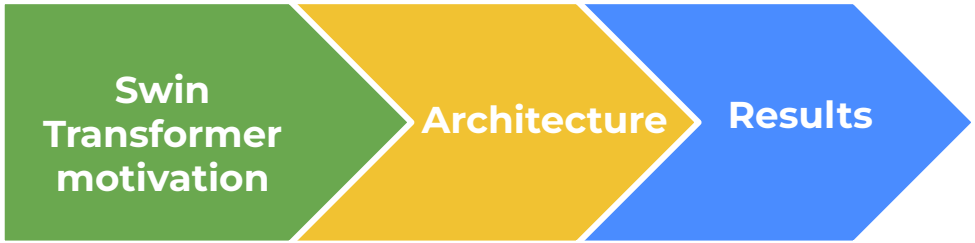


Image Source: Swim Transformers

In This Chapter



Swin Transformer Architecture

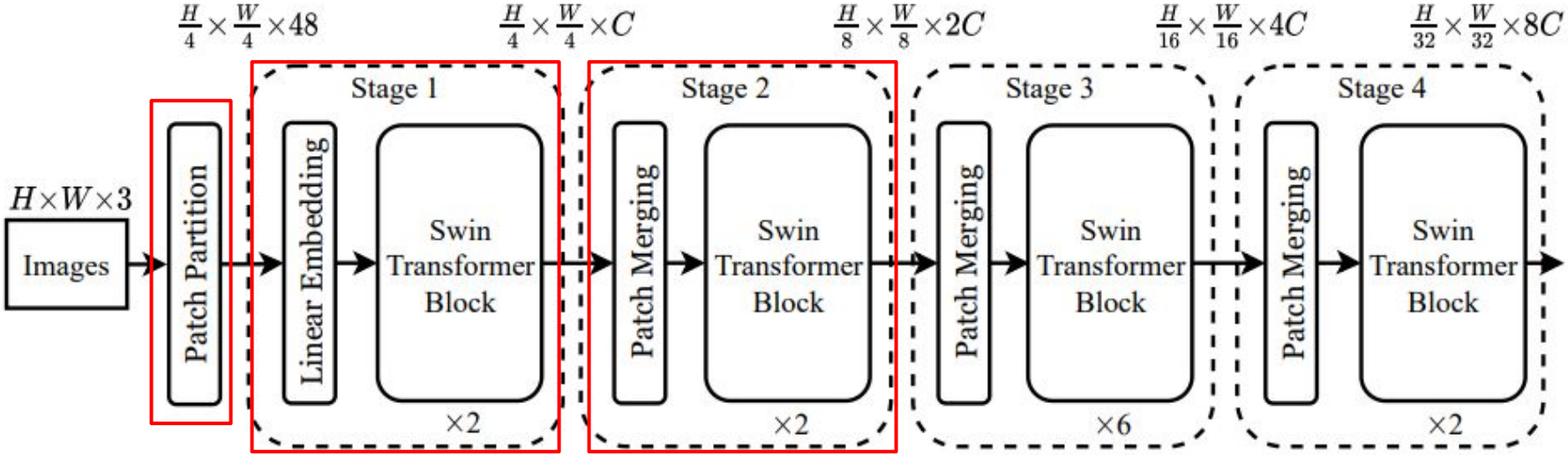


Image Source: Swim Transformers

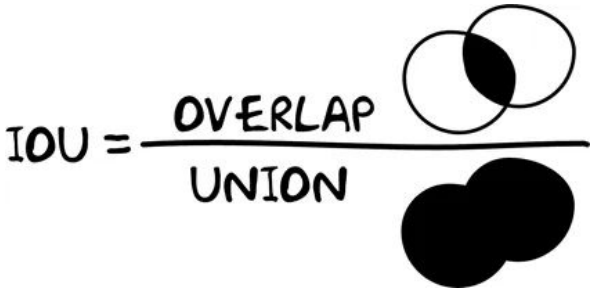
In This Chapter

Classification Results

(b) ImageNet-22K pre-trained models					
method	image size	#param.	FLOPs	throughput (image / s)	ImageNet top-1 acc.
R-101x3 [38]	384^2	388M	204.6G	-	84.4
R-152x4 [38]	480^2	937M	840.5G	-	85.4
ViT-B/16 [20]	384^2	86M	55.4G	85.9	84.0
ViT-L/16 [20]	384^2	307M	190.7G	27.3	85.2
Swin-B	224^2	88M	15.4G	278.1	85.2
Swin-B	384^2	88M	47.0G	84.7	86.4
Swin-L	384^2	197M	103.9G	42.1	87.3

Image Source: Swim Transformers

IoU



Object detection

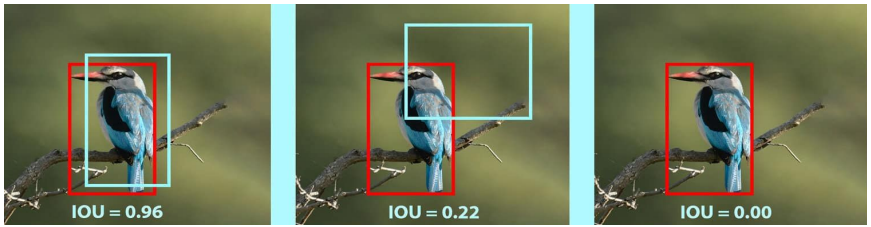


Image Segmentation

Image Source: Swim Transformers

Segmentation Results

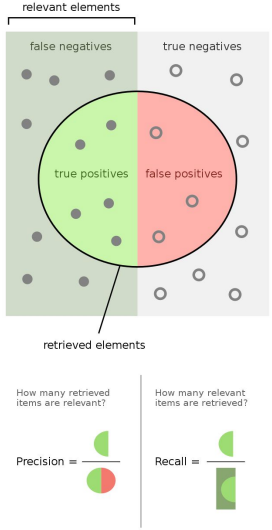
ADE20K Method	Backbone	val mIoU	test score	#param.	FLOPs	FPS
DANet [23]	ResNet-101	45.2	-	69M	1119G	15.2
DLab.v3+ [11]	ResNet-101	44.1	-	63M	1021G	16.0
ACNet [24]	ResNet-101	45.9	38.5	-		
DNL [71]	ResNet-101	46.0	56.2	69M	1249G	14.8
OCRNet [73]	ResNet-101	45.3	56.0	56M	923G	19.3
UperNet [69]	ResNet-101	44.9	-	86M	1029G	20.1
OCRNet [73]	HRNet-w48	45.7	-	71M	664G	12.5
DLab.v3+ [11]	ResNeSt-101	46.9	55.1	66M	1051G	11.9
DLab.v3+ [11]	ResNeSt-200	48.4	-	88M	1381G	8.1
SETR [81]	T-Large‡	50.3	61.7	308M	-	-
UperNet	DeiT-S†	44.0	-	52M	1099G	16.2
UperNet	Swin-T	46.1	-	60M	945G	18.5
UperNet	Swin-S	49.3	-	81M	1038G	15.2
UperNet	Swin-B‡	51.6	-	121M	1841G	8.7
UperNet	Swin-L‡	**53.5**	**62.8**	234M	3230G	6.2

Image Source: Swim Transformers

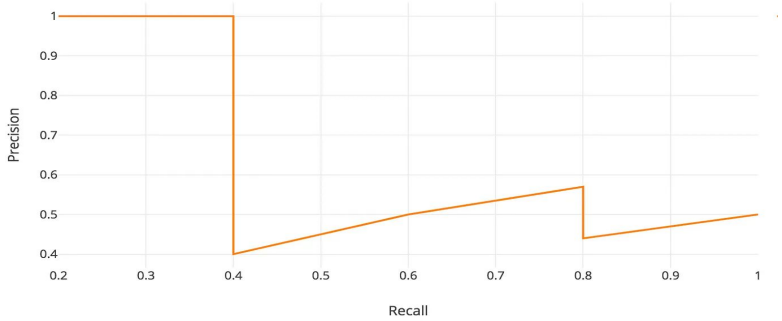
AP - Average Precision

Confidence rank Table

Rank	Correct?	Precision	Recall
1	True	1.0 ↑	0.2 ↑
2	True	1.0 –	0.4 ↑
3	False	0.67 ↓	0.4 –
4	False	0.5 ↓	0.4 –
5	False	0.4 ↓	0.4 –
6	True	0.5 ↑	0.6 ↑
7	True	0.57 ↑	0.8 ↑



Precision - Recall



Smooth Precision - Recall

Detection Results

(c) System-level Comparison						
Method	mini-val		test-dev		#param.	FLOPs
	AP^{box}	AP^{mask}	AP^{box}	AP^{mask}		
RepPointsV2* [12]	-	-	52.1	-	-	-
GCNet* [7]	51.8	44.7	52.3	45.4	-	1041G
RelationNet++* [13]	-	-	52.7	-	-	-
SpineNet-190 [21]	52.6	-	52.8	-	164M	1885G
ResNeSt-200* [78]	52.5	-	53.3	47.1	-	-
EfficientDet-D7 [59]	54.4	-	55.1	-	77M	410G
DetectoRS* [46]	-	-	55.7	48.5	-	-
YOLOv4 P7* [4]	-	-	55.8	-	-	-
Copy-paste [26]	55.9	47.2	56.0	47.4	185M	1440G
X101-64 (HTC++)	52.3	46.0	-	-	155M	1033G
Swin-B (HTC++)	56.4	49.1	-	-	160M	1043G
Swin-L (HTC++)	57.1	49.5	57.7	50.2	284M	1470G
Swin-L (HTC++)*	**58.0**	**50.4**	**58.7**	**51.1**	284M	-

Image Source: Swim Transformers

Questions?

04

Conclusion

Conclusion

Novelty

The birth of Attention developed into Transformers has brought a new era to ML

Performance

Transformers are in many cases replacing CNNs and RNNs - the most popular types of deep learning models just five years ago

Versatile

Can be easily used in transfer learning with little to no fine-tuning on specific tasks

Future to come

70% percent of arXiv papers on AI posted in the last two years mention transformers

Thank you!