Dauphine | PSL
UNIVERSITÉ PARIS

Deep Learning for Image Analysis

Mehyar MLAWEH

🏫 Master BDIA - Dauphine Tunis
PhD student - Université PSL

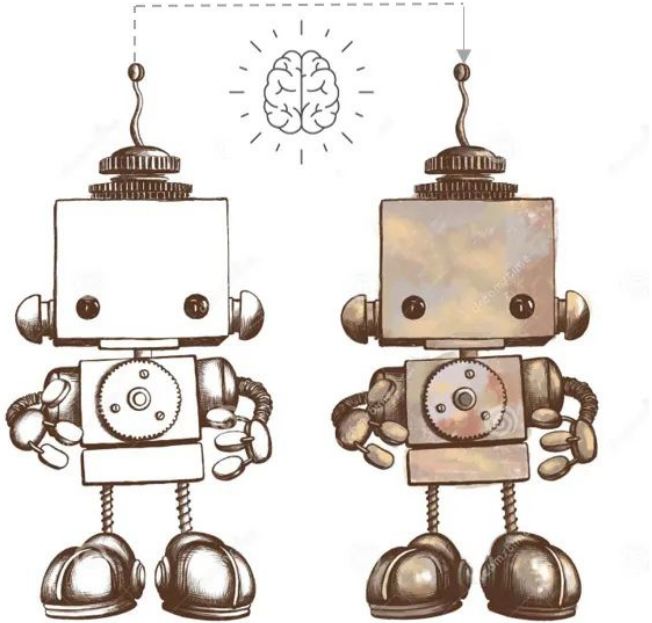
Resources: https://mehyarmlaweh.github.io/mehyar-mlaweh/teaching

Module Overview

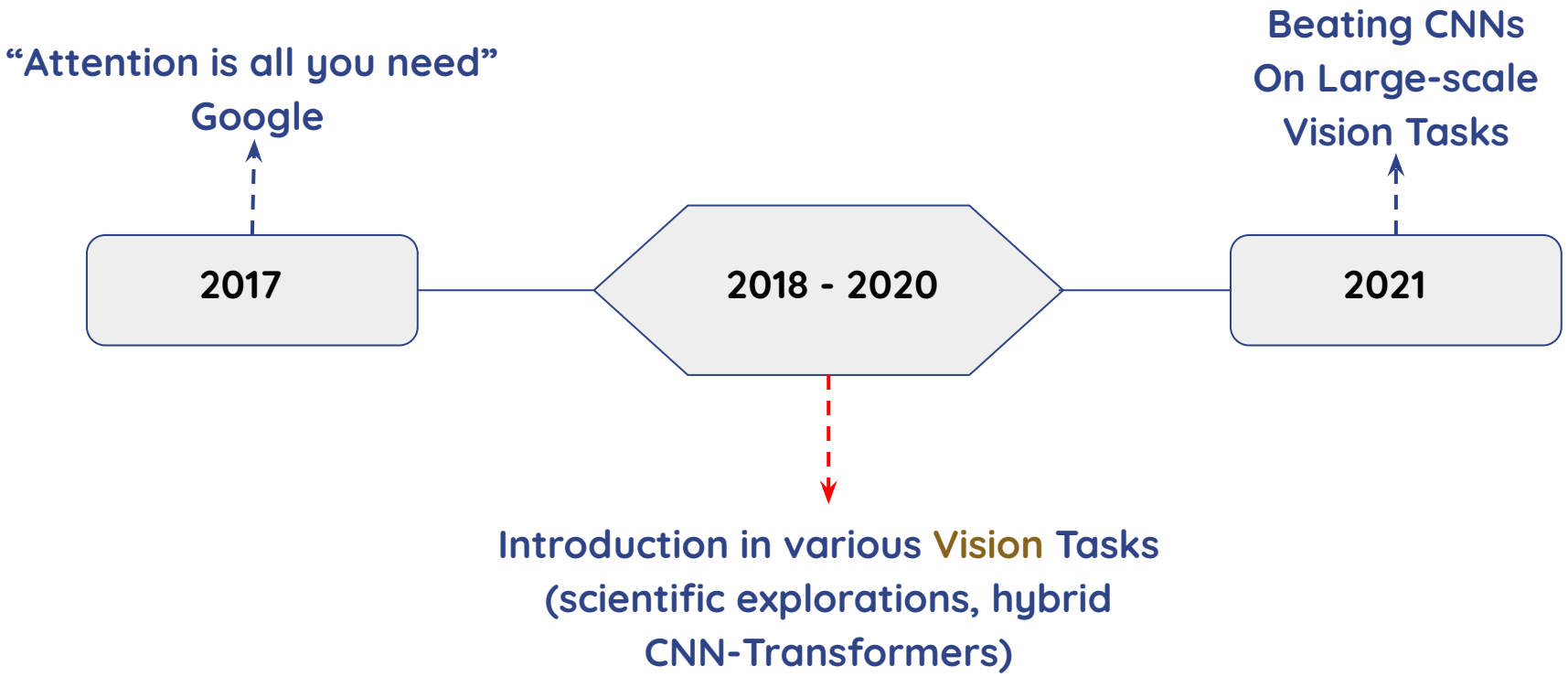
1. Computer Vision Foundations
2. CNNs
3. Transfer Learning
4. Vision Transformers
5. Object Detection
6. Segment Anything Model (SAM)
7. Final Project

Lecture 4 :
Vision Transformers

Last Time



History of Transformers

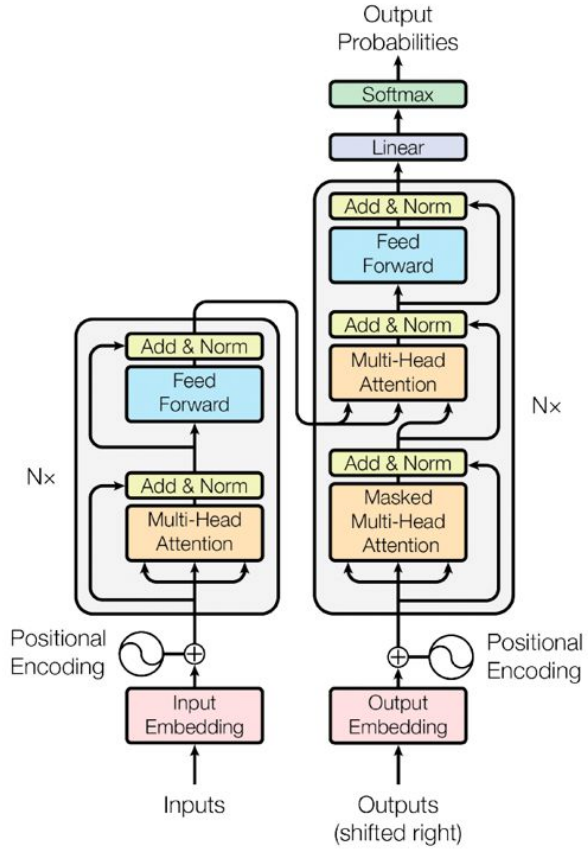


Transformers

THE TRANSFORMER

Je suis étudiant →  → I am a student

Transformers

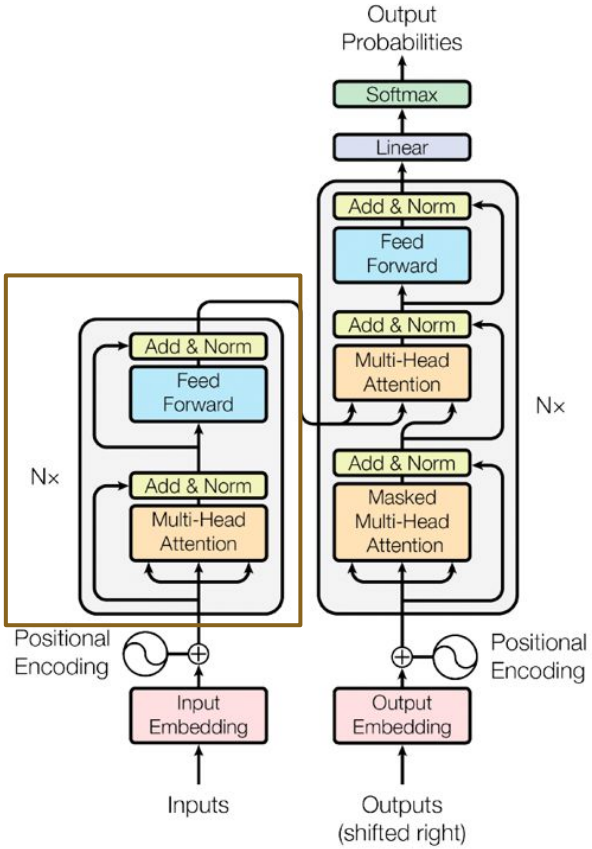


THE TRANSFORMER

Je suis étudiant →  → I am a student

Transformers

Encoders



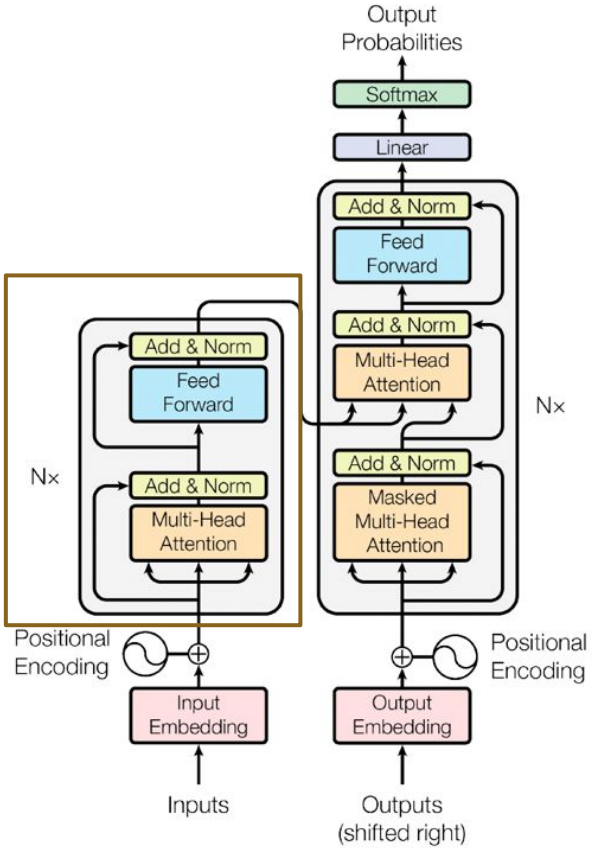
THE TRANSFORMER

Je suis étudiant →  → I am a student

Transformers

Encoders

N identical encoders stacked
(N = 6 in the original paper)



THE TRANSFORMER

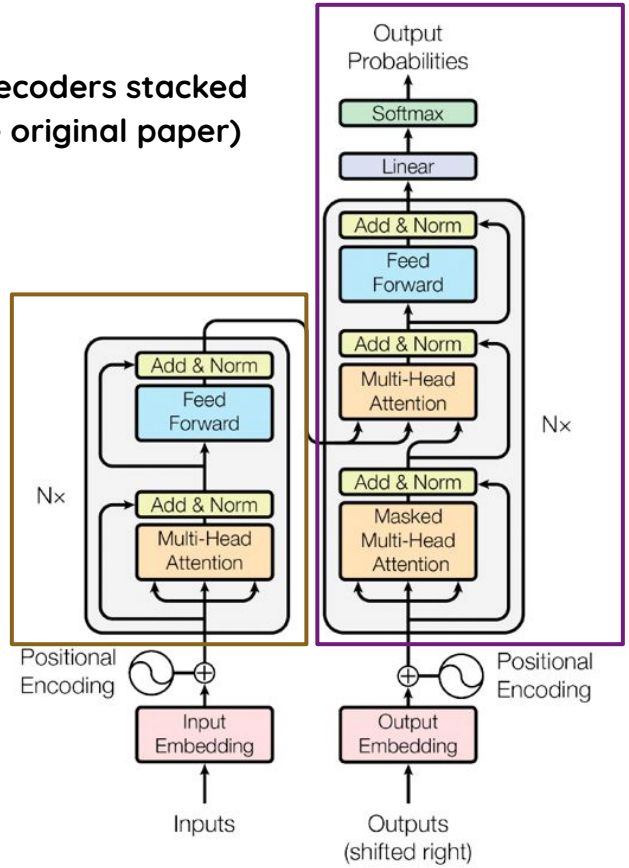
Transformers

N identical decoders stacked
(N = 6 in the original paper)

Decoders

Encoders

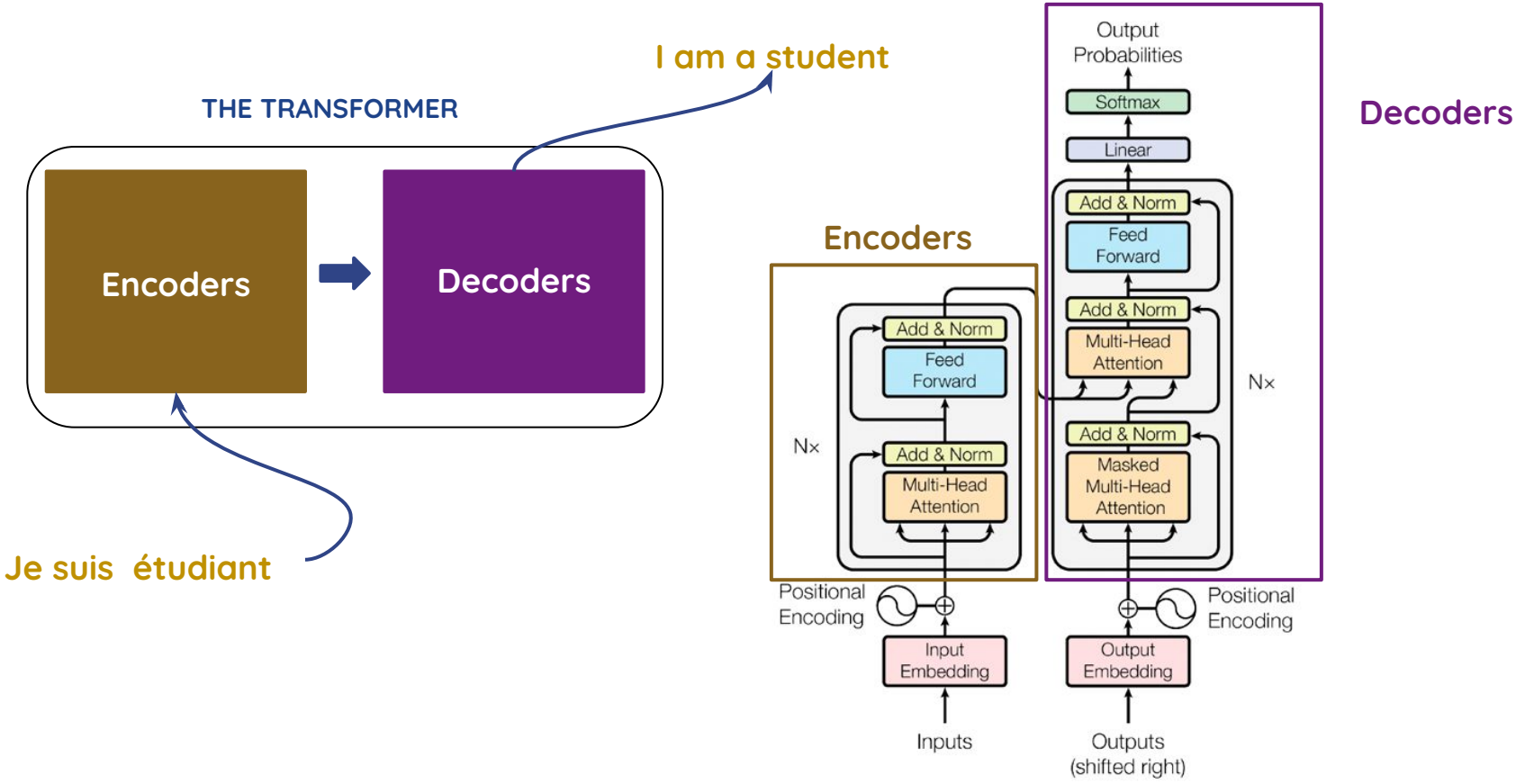
N identical encoders stacked
(N = 6 in the original paper)



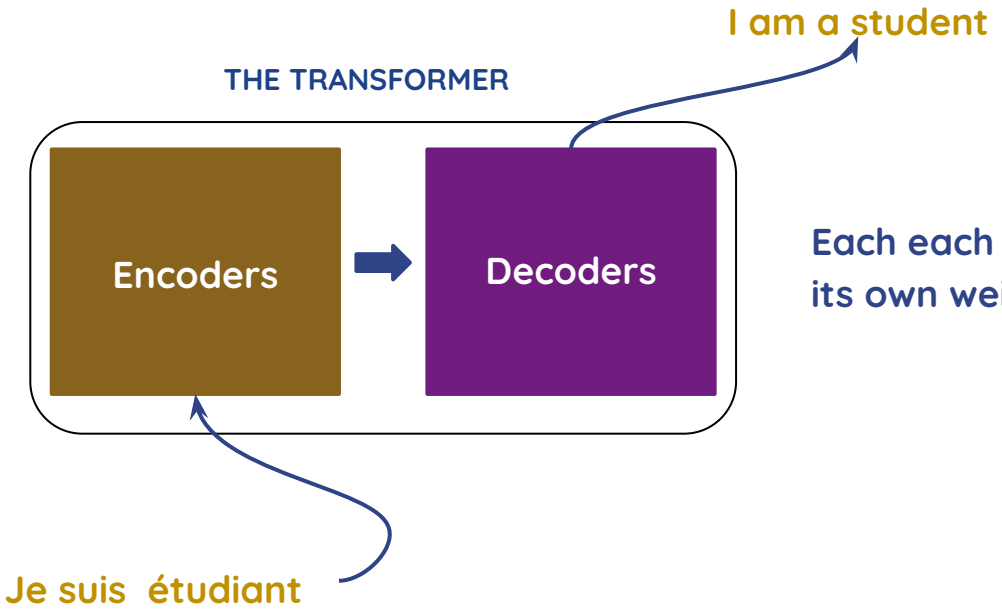
THE TRANSFORMER

Je suis étudiant →  → I am a student

Transformers

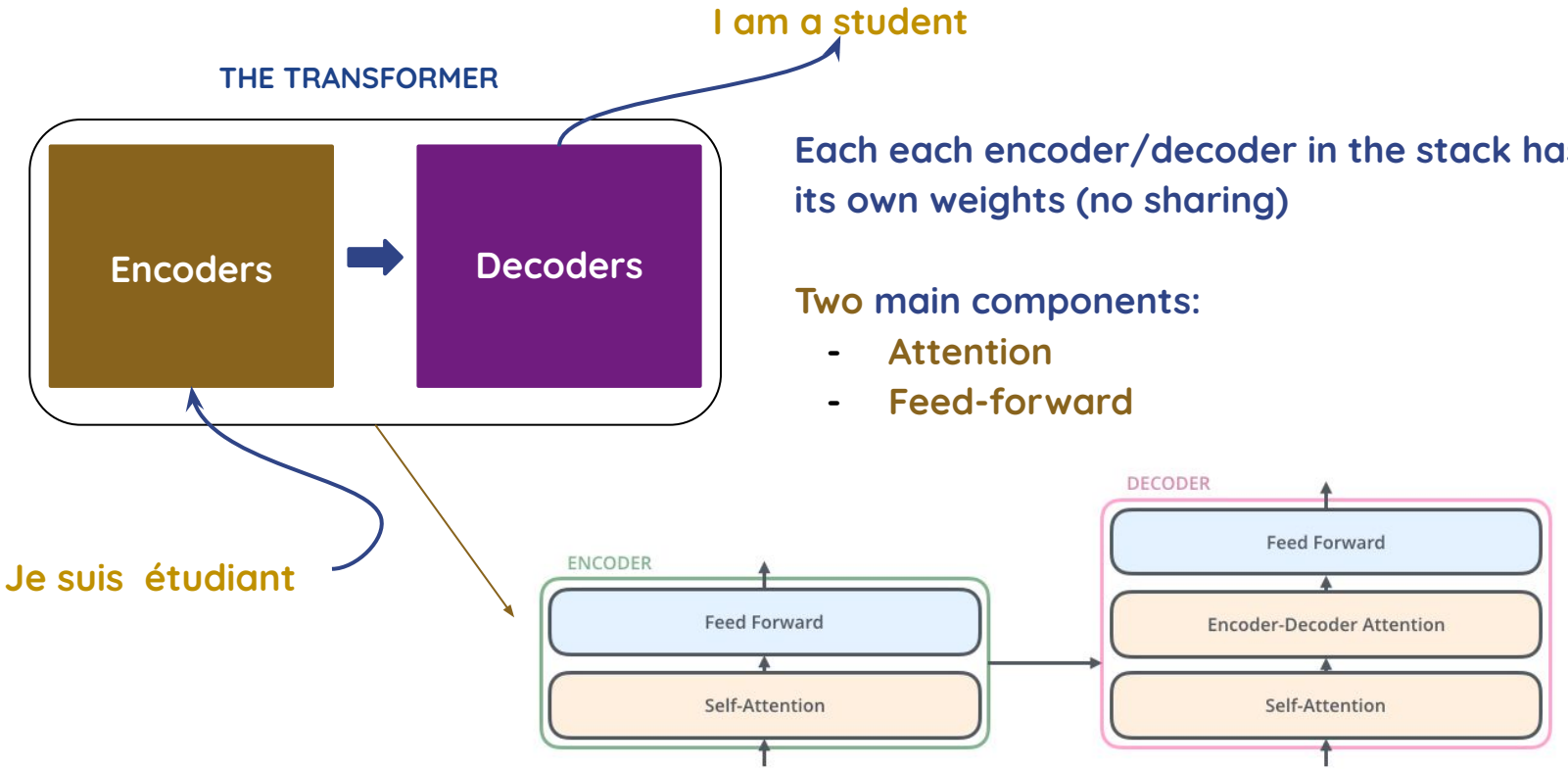


Transformers



Each each encoder/decoder in the stack has its own weights (no sharing)

Transformers



Each each encoder/decoder in the stack has its own weights (no sharing)

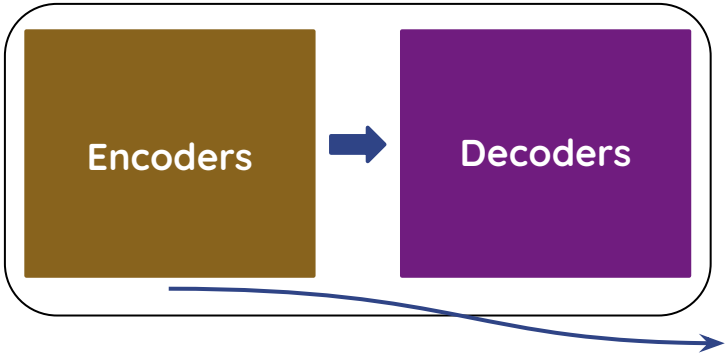
Two main components:

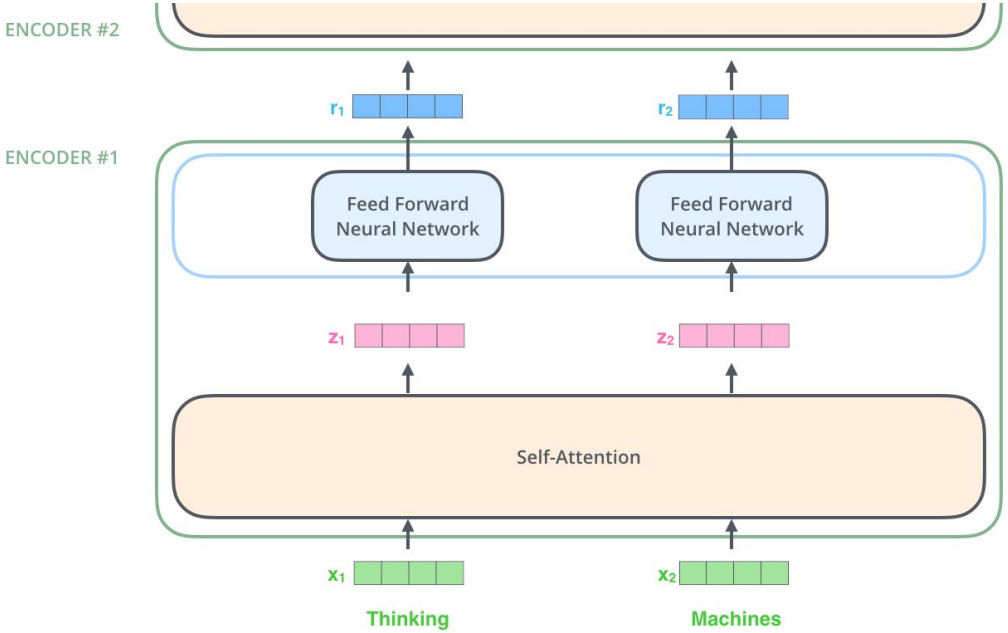
- Attention
- Feed-forward

[Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.]

Transformers

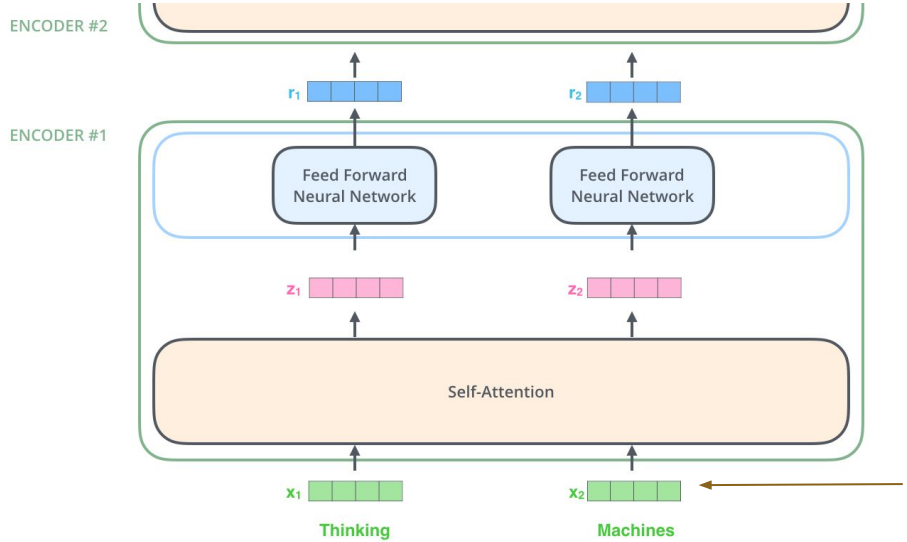
THE TRANSFORMER





[Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.]

Transformers - Input



Step 1.1 : Input Preparation

Words in the input sequence are converted into numerical representations (vectors) using embedding techniques.

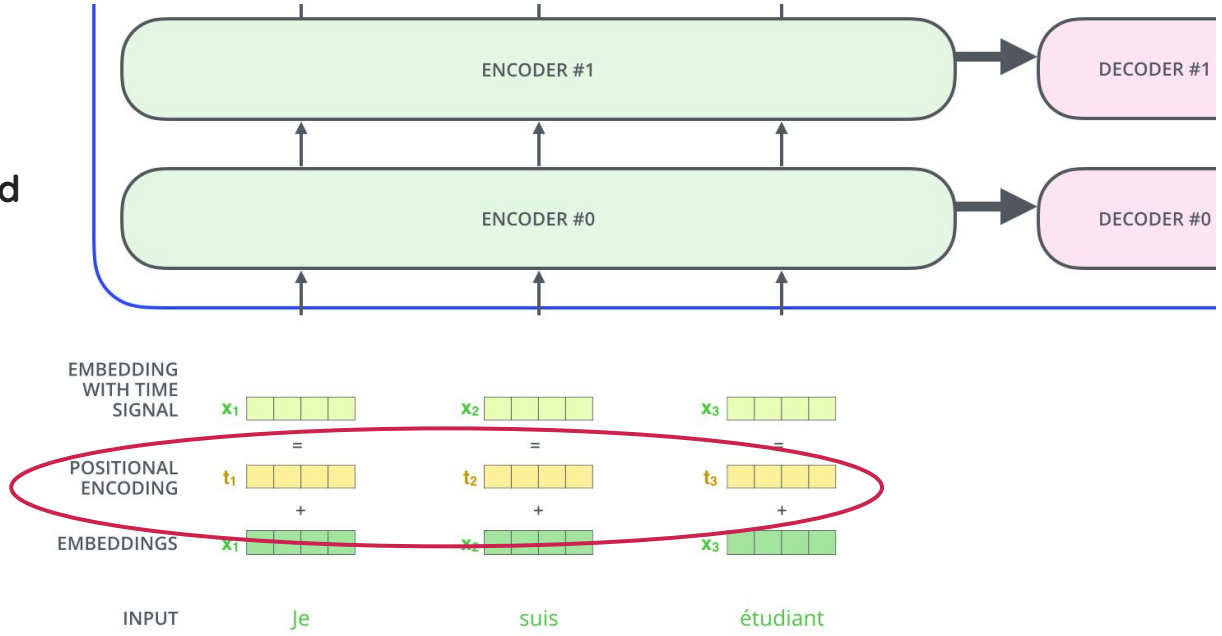
These vectors are not random, they are learned during training

[Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.]

Transformers - Input

Step 1.2 : Positional encoding

To have a clue about order and position of each word, the transformer adds a vector to each input embedding

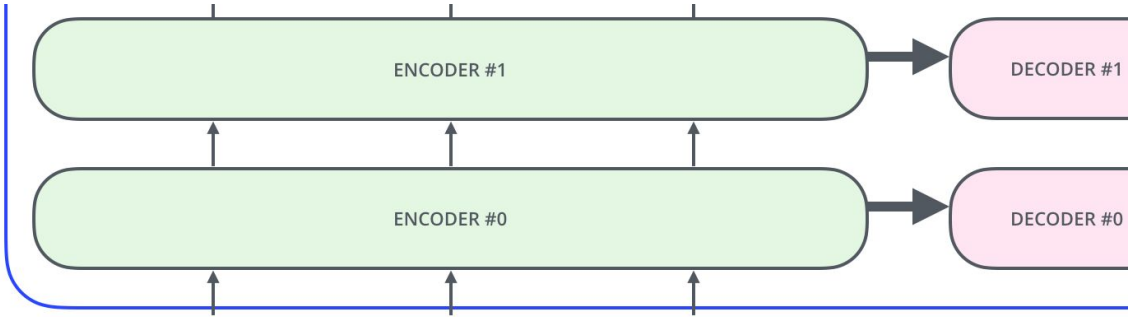


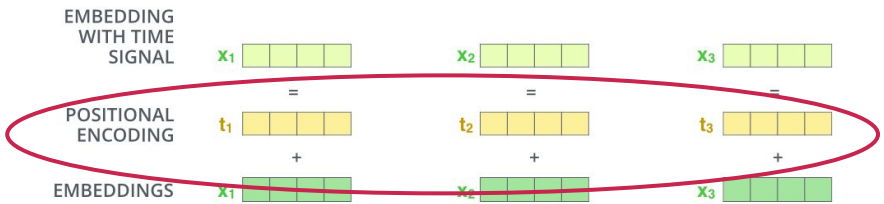
[Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.]

Transformers - Input

Step 1.2 : Positional encoding

To have a clue about order and position of each word, the transformer adds a vector to each input embedding

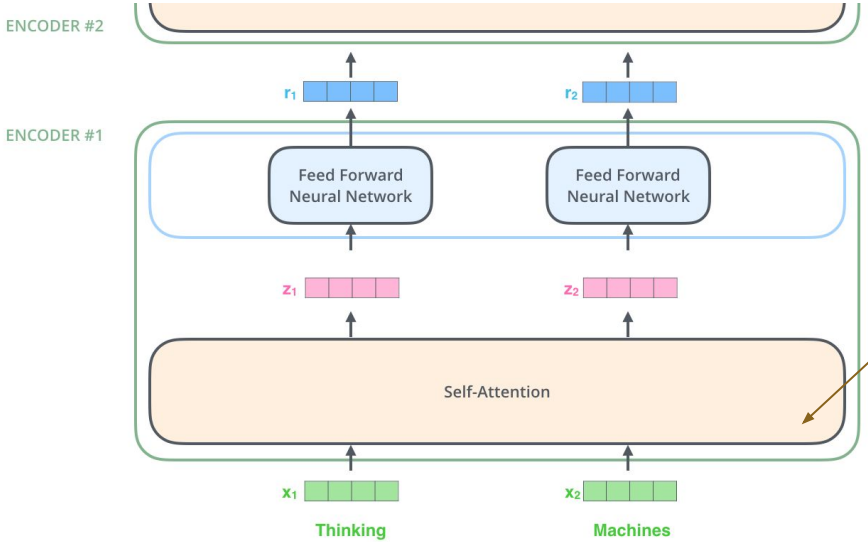




Example:

[Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.]

Transformers - Encoder Layer

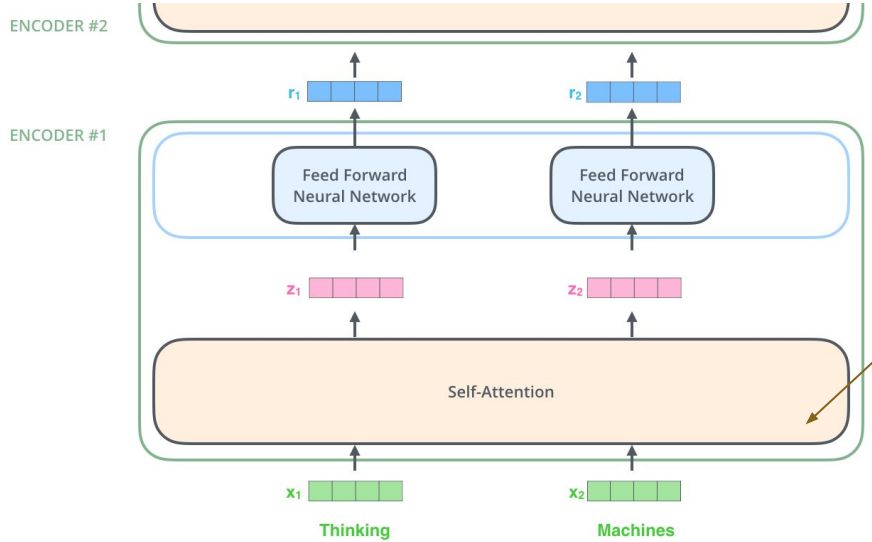


Step 2 : Self-Attention

Analyzes the relationships between each word in the sequence, allowing the model to understand how words depend on each other.

[Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.]

Transformers - Encoder Layer

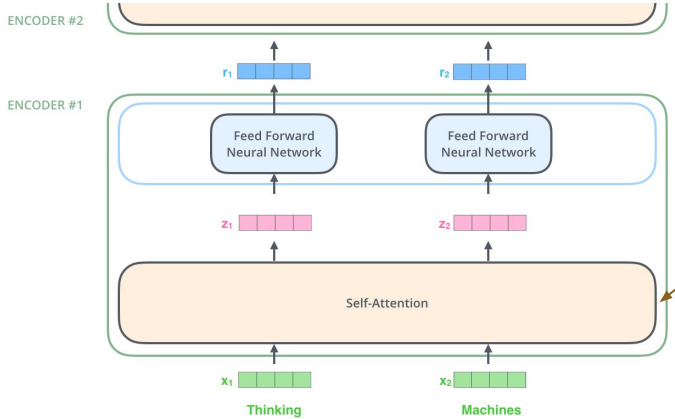


Step 2 : Self-Attention

Which words are important for me?

[Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.]

Transformers - Encoder Layer



Step 2 : Self-Attention

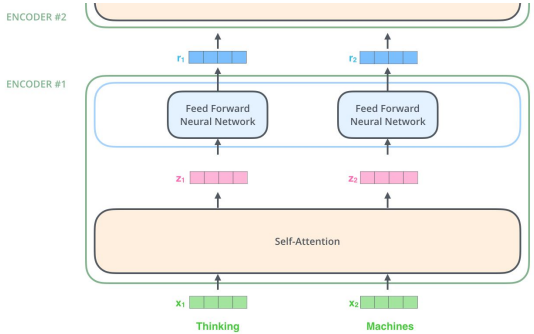
Example

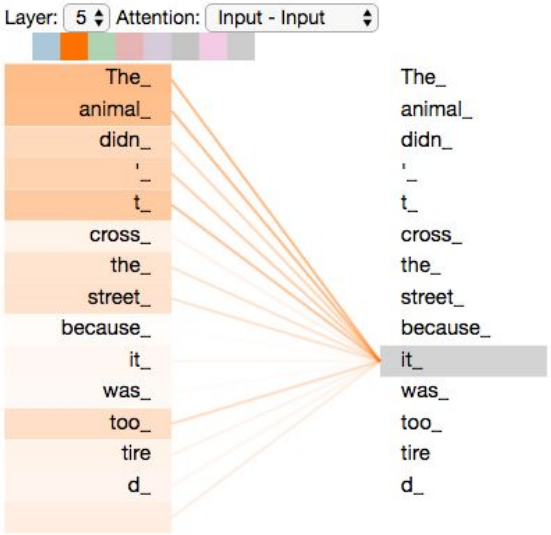
"The animal didn't cross the street because **it** was too tired"

What does 'it' refer to?

[Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.]

Transformers - Encoder Layer





Step 2 : Self-Attention:

Example

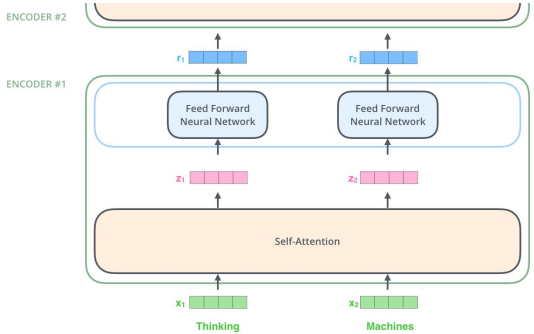
"The animal didn't cross the street because **it** was too tired"

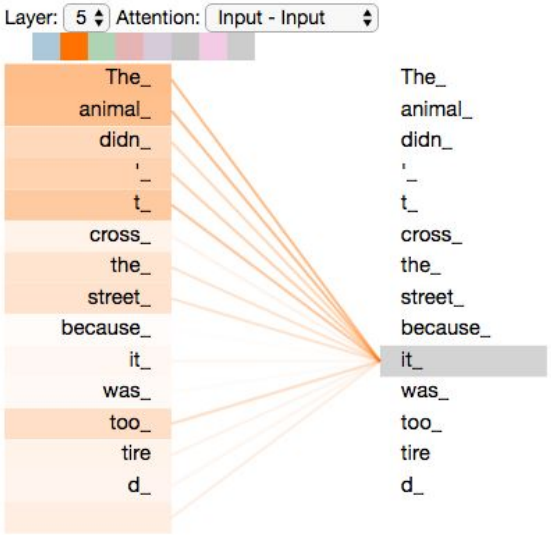
What does 'it' refer to?

We compute the attention for this specific word

[Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.]

Transformers - Encoder Layer





Step 2 : Self-Attention:

Example

"The animal didn't cross the street because **it** was too tired"

What does 'it' refer to?

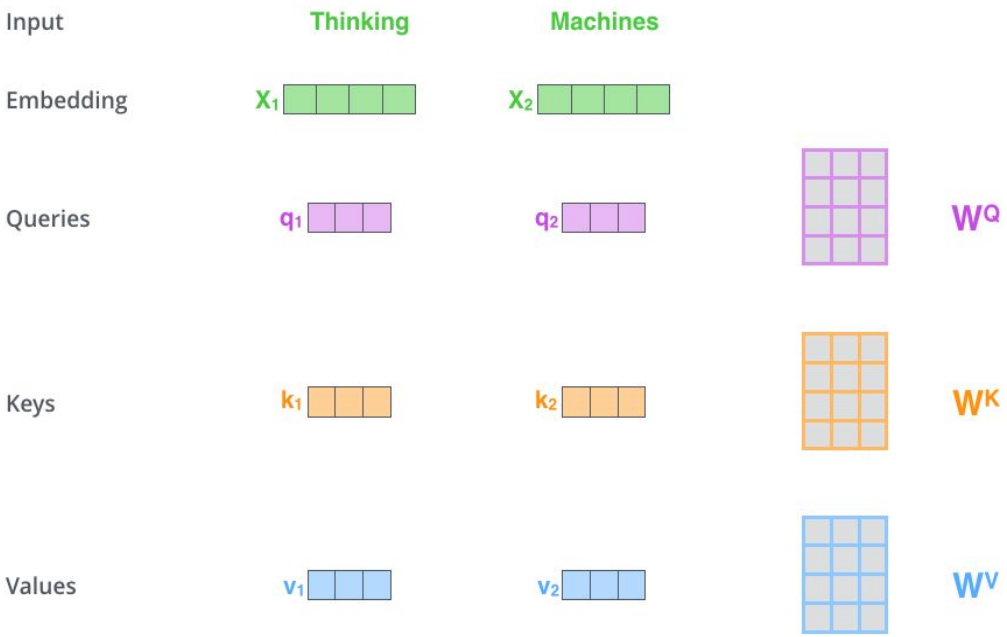
We compute the attention for this specific word

-> Attention scores indicate which tokens are the most relevant for the given output

[Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.]

Transformers - Encoder Layer

Self-Attention: Query, Key, Value



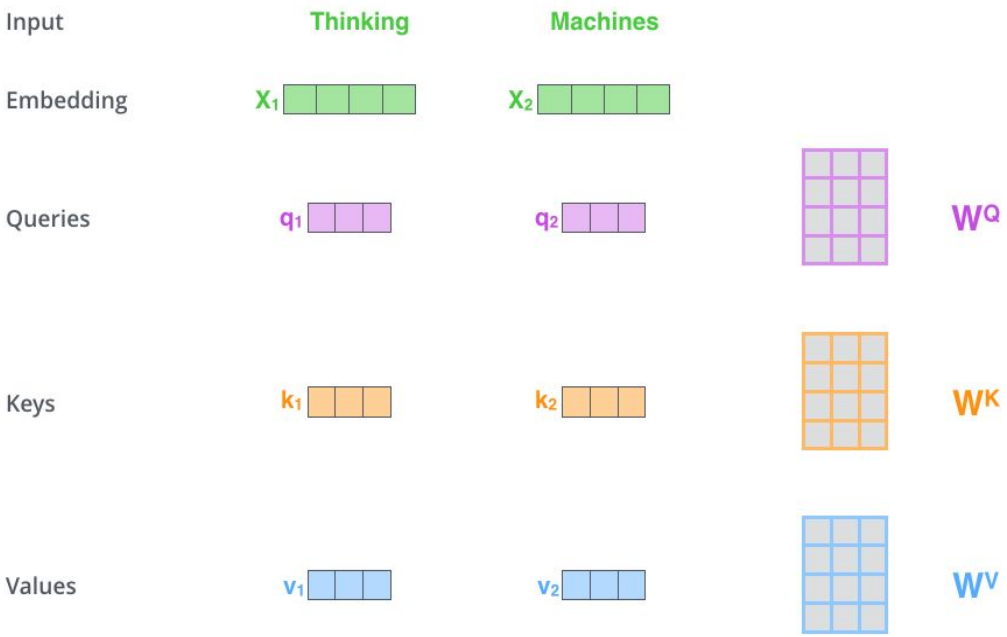
Step 2.1

So for each word, we create a Q vector, a K vector, and a V vector

[Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.]

Transformers - Encoder Layer

Self-Attention: Query, Key, Value



Step 2.1

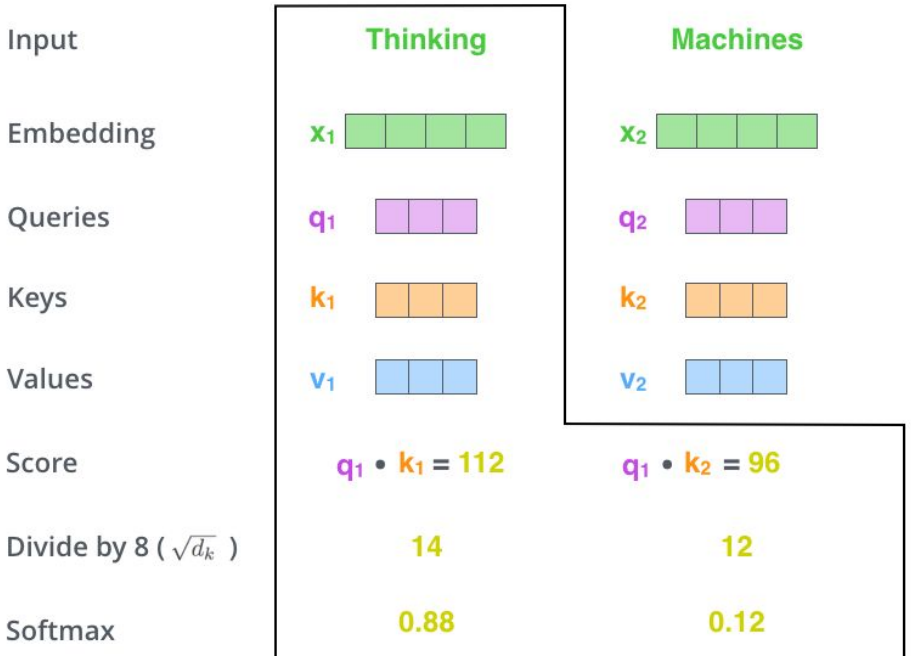
So for each word, we create a Q vector, a K vector, and a V vector

These vectors are created by multiplying the embedding by the three matrices that we trained during the training process.

[Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.]

Transformers - Encoder Layer

Self-Attention: Query, Key, Value



[Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.]

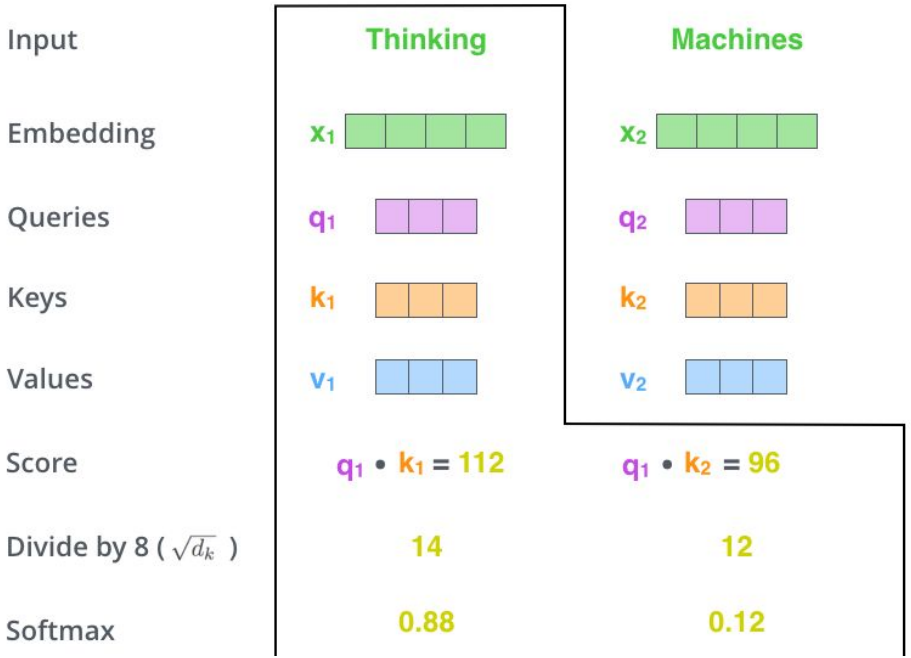
Step 2.2

Compute attention scores & normalize.

We compute attention scores by taking the dot product between a query and all keys

Transformers - Encoder Layer

Self-Attention: Query, Key, Value



[Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.]

Step 2.2

Compute attention scores & normalize.

We compute attention scores by taking the dot product between a query and all keys

For token 1:

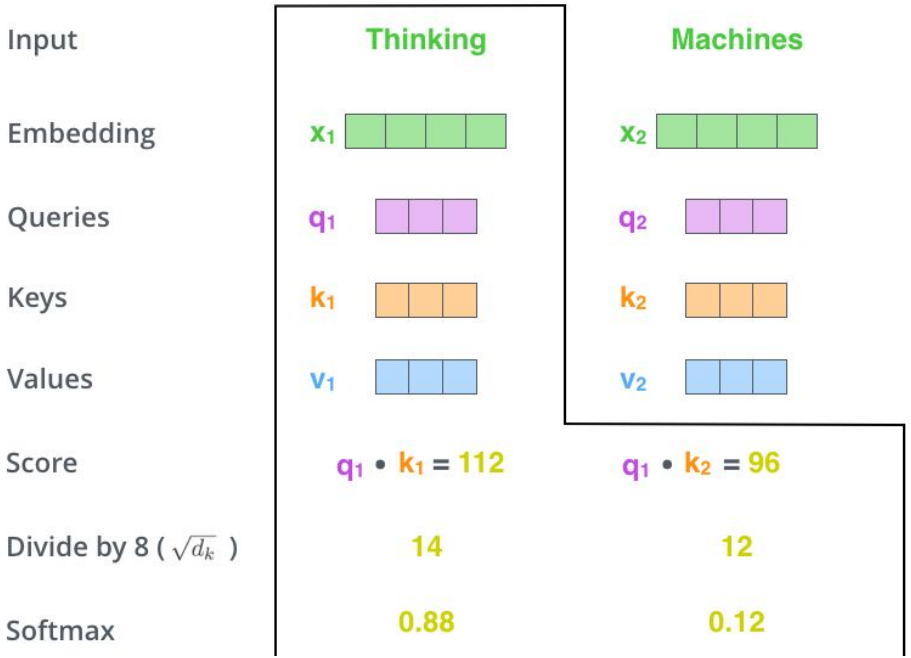
- Score with itself: $q1 \cdot k1$
- Score with token 2: $q1 \cdot k2$

Larger dot product → higher relevance.

Which word should "Thinking" attend to more? Why?

Transformers - Encoder Layer

Self-Attention: Query, Key, Value



[Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.]

Step 2.2

Compute attention scores & normalize.

We compute attention scores by taking the dot product between a query and all keys

For token 1:

- Score with itself: $q1 \cdot k1$
- Score with token 2: $q1 \cdot k2$

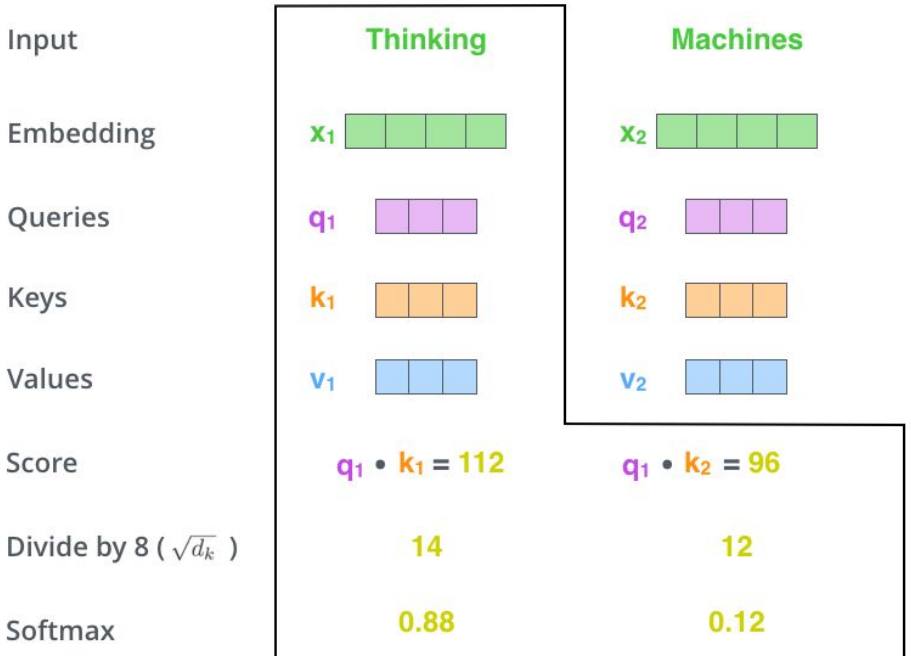
Larger dot product → higher relevance.

112 > 96

The word Thinking : "*I already carry useful information :)* "

Transformers - Encoder Layer

Self-Attention: Query, Key, Value



[Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.]

Step 2.2

Compute attention scores & normalize.

We compute attention scores by taking the dot product between a query and all keys

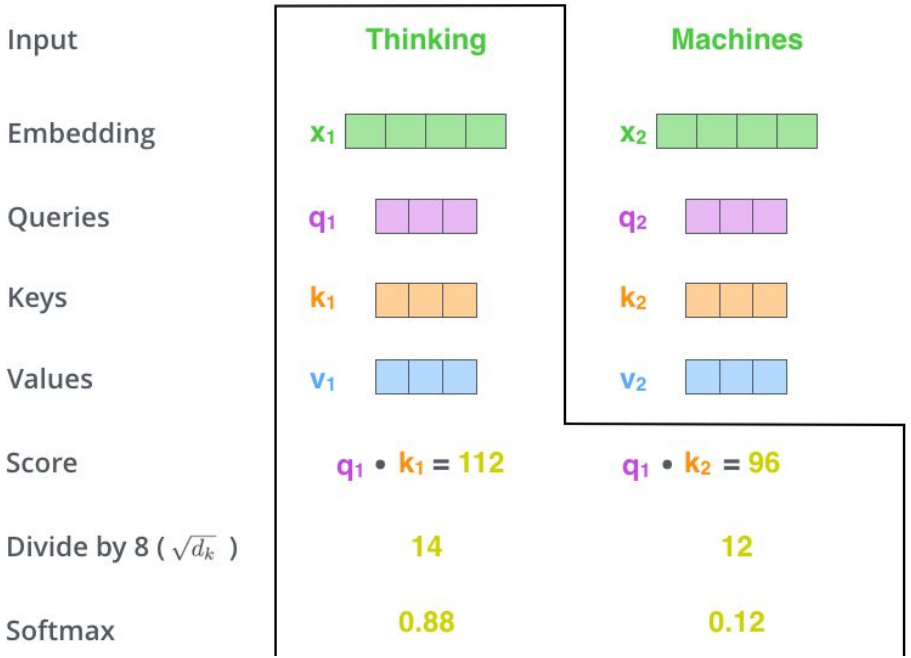
For token 1:

- Score with itself: $q1 \cdot k1$
- Score with token 2: $q1 \cdot k2$

The dot product can become very large, especially with high-dimensional vectors

Transformers - Encoder Layer

Self-Attention: Query, Key, Value



[Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.]

Step 2.2

Compute attention scores & normalize.

We compute attention scores by taking the dot product between a query and all keys

For token 1:

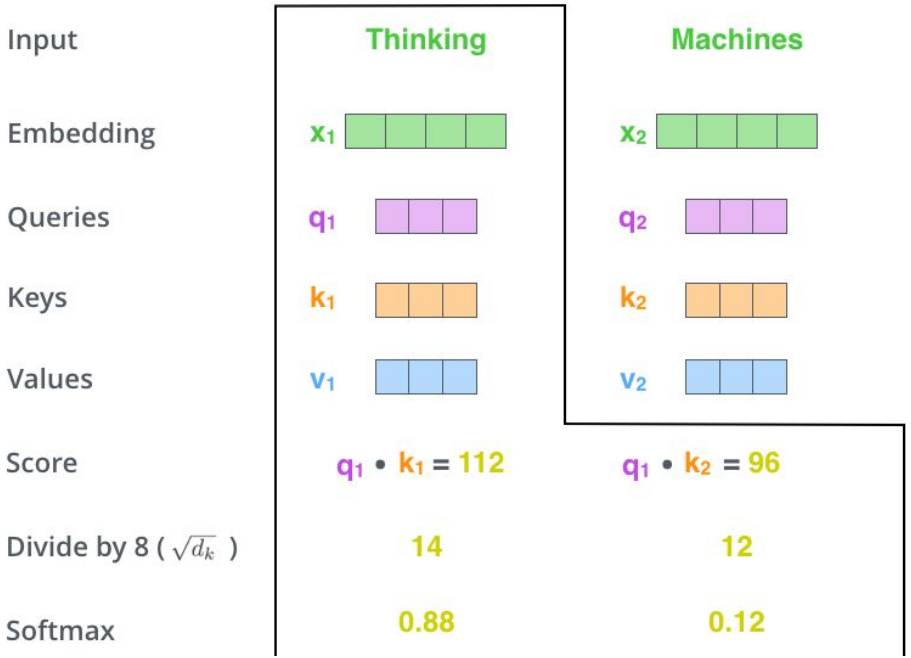
- Score with itself: $q1 \cdot k1$
- Score with token 2: $q1 \cdot k2$

The dot product can become very large, especially with high-dimensional vectors

So we divide by √d to keep values in a reasonable range

Transformers - Encoder Layer

Self-Attention: Query, Key, Value



[Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.]

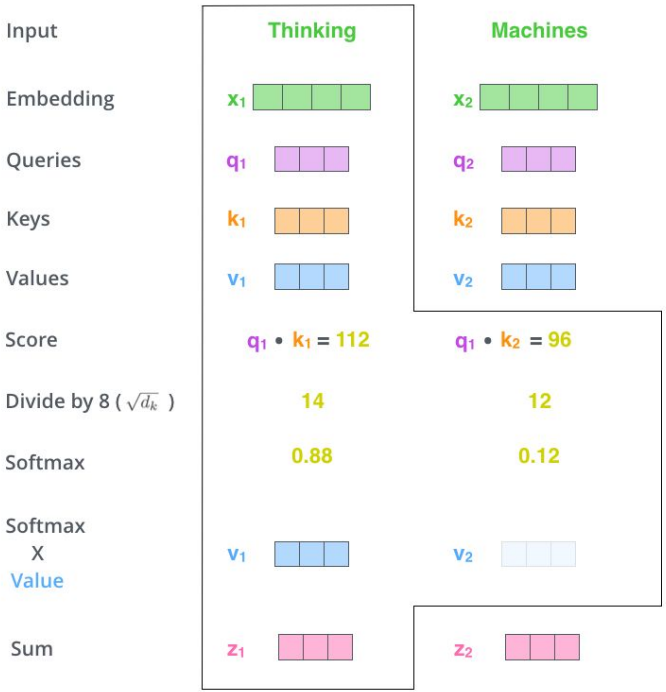
Step 2.3

Softmax

Softmax turns raw scores into attention weights (probabilities)

Transformers - Encoder Layer

Self-Attention: Query, Key, Value



[Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.]

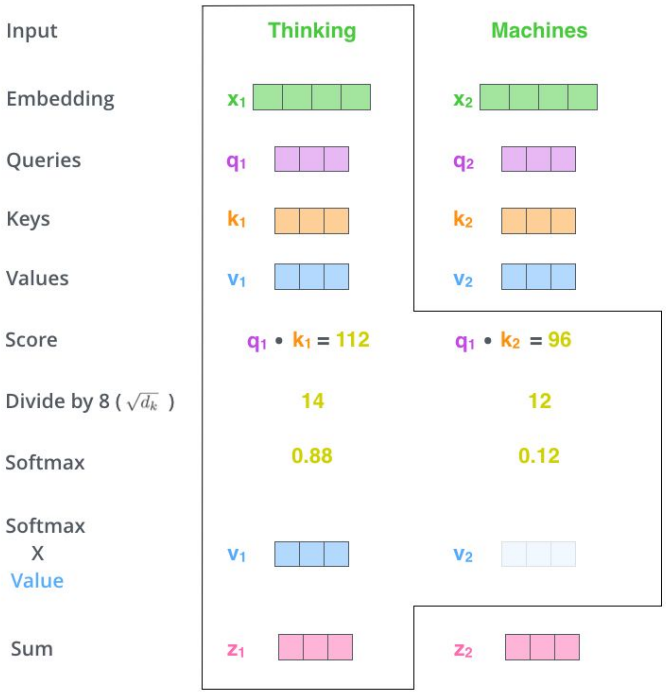
Step 2.4

Apply weights to values & sum

Now we multiply each value by its attention weight

Transformers - Encoder Layer

Self-Attention: Query, Key, Value



[Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.]

Step 2.4

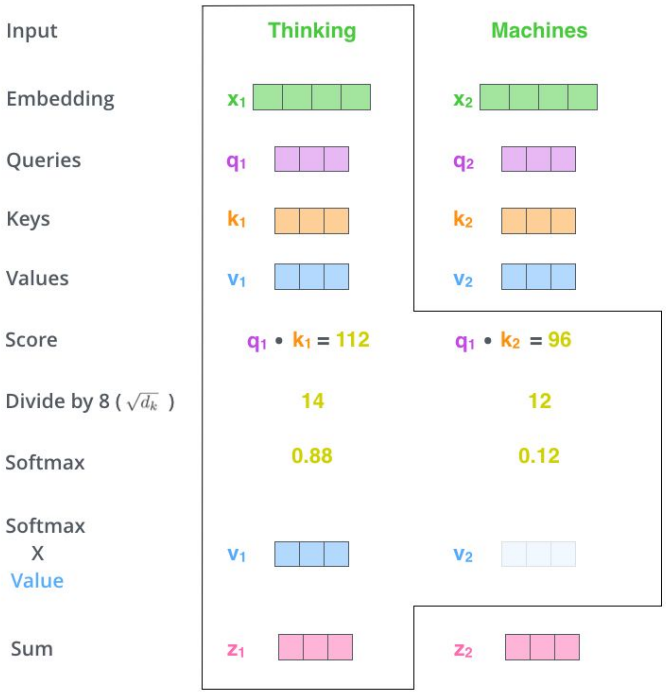
Apply weights to values & sum

Now we multiply each value by its attention weight

Finally, we sum everything to get the new representation z1

Transformers - Encoder Layer

Self-Attention: Query, Key, Value



[Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.]

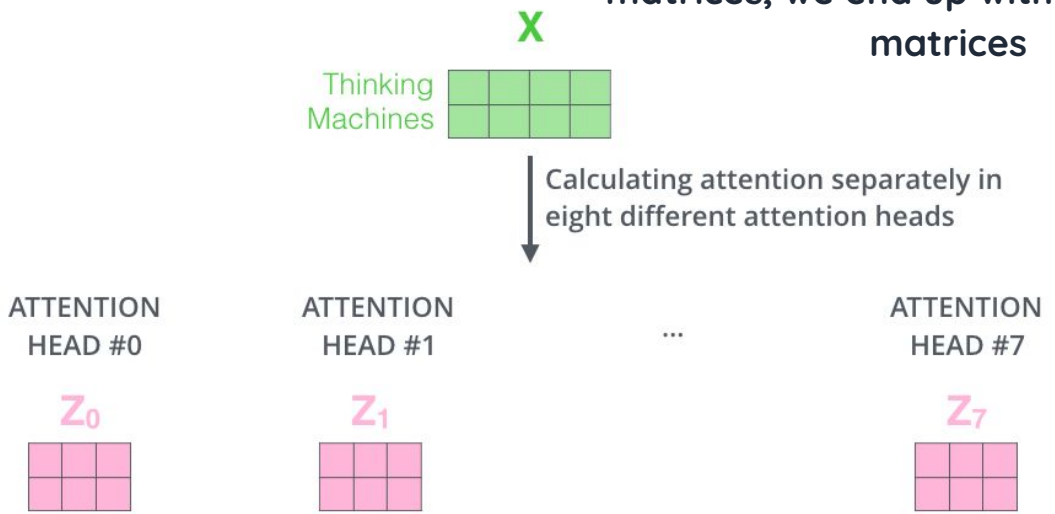
We just computed the output for ONE word

We repeat this for every word in parallel

Transformers - Encoder Layer

Self-Attention: Query, Key, Value

8 different times with different weight matrices, we end up with 8 different Z matrices



[Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.]

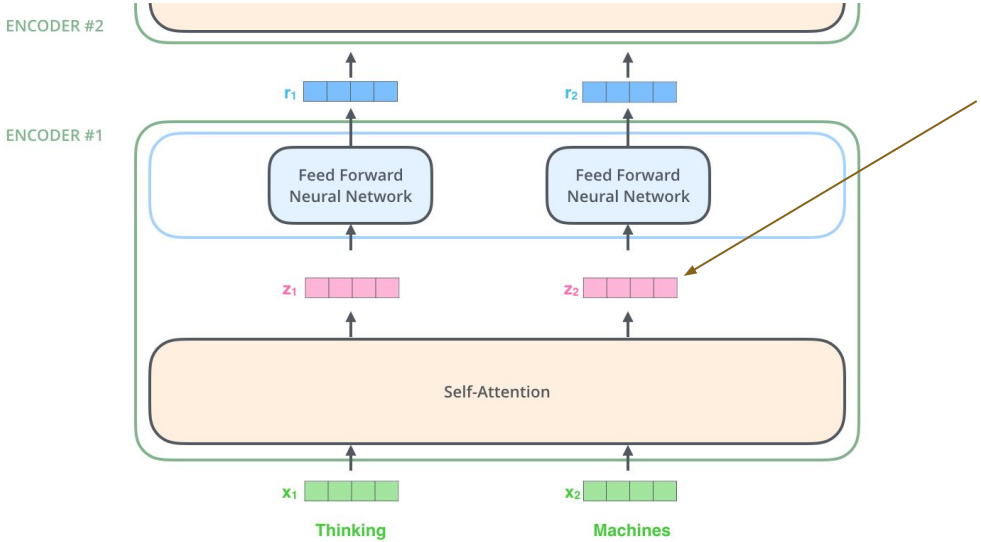
PS: Original Transformer (Vaswani et al., 2017) : 8 heads

Transformers - Encoder Layer

Self-Attention: Query, Key, Value

Interactive: Visualizing Attention

Transformers - Encoder Layer

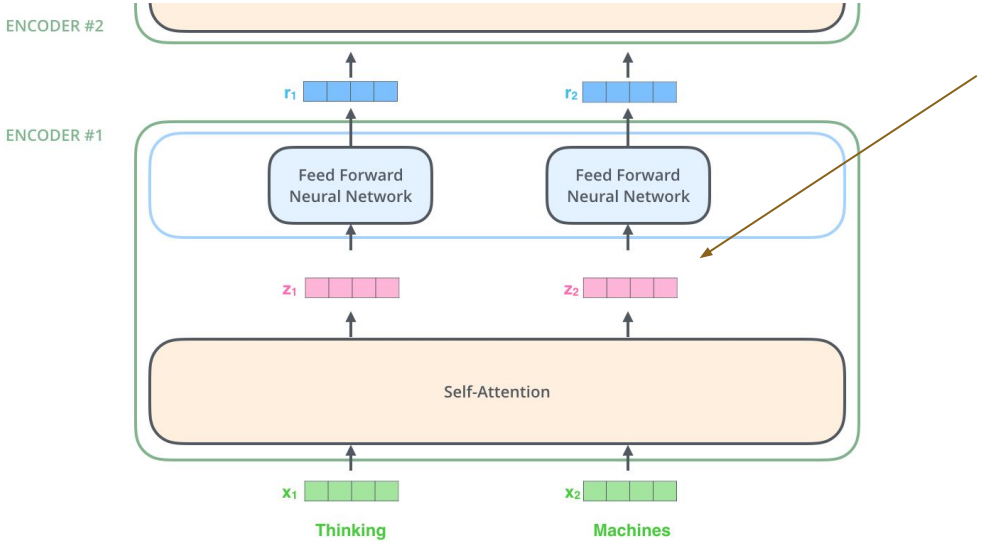


Step 3

Concatenate all the attention head

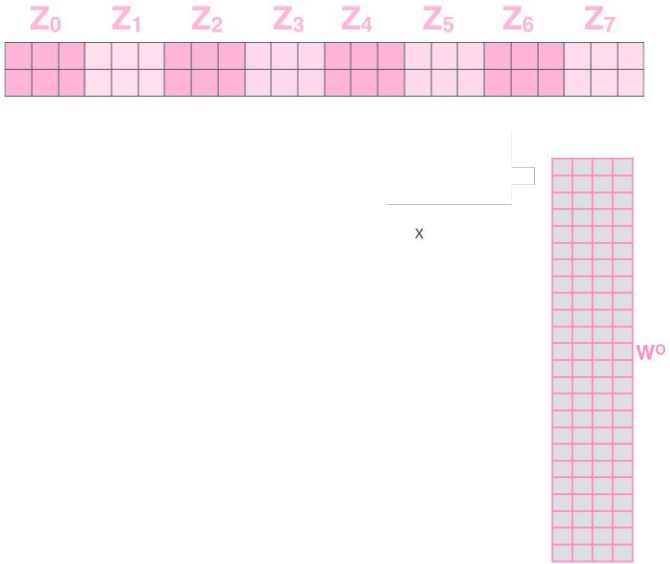
[Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.]

Transformers - Encoder Layer



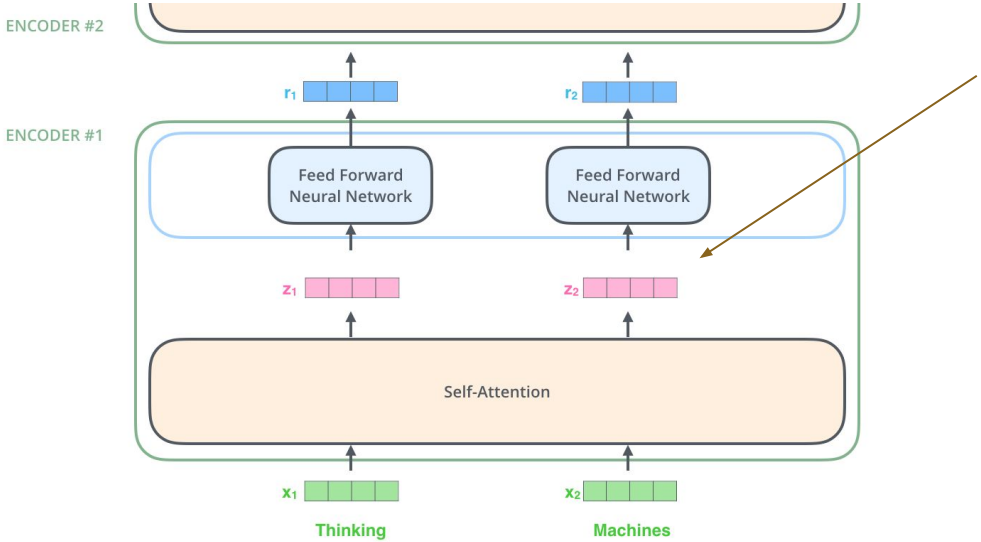
Step 3

Multiply with a weigh matrix W0 that was trained with the model



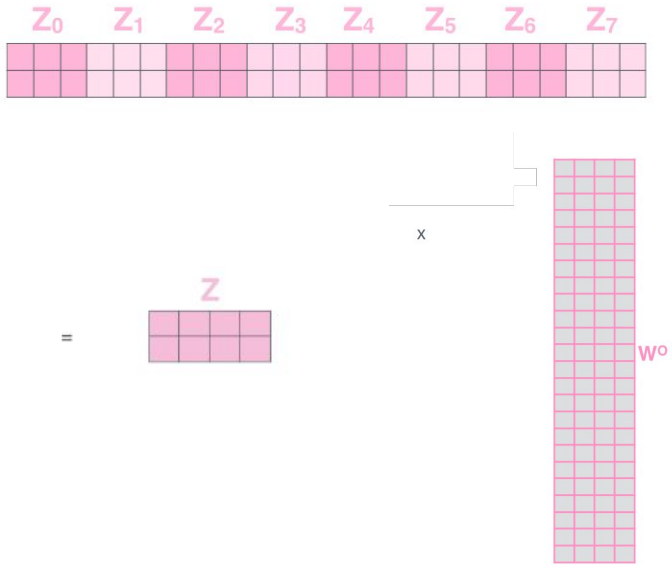
[Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.]

Transformers - Encoder Layer



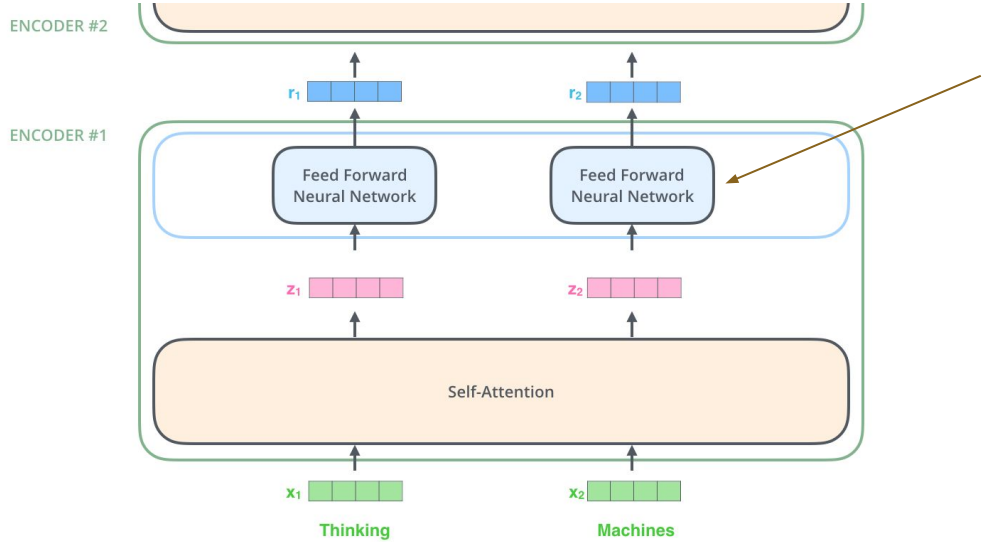
Step 3

Result would be the Z matrix



[Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.]

Transformers - Encoder Layer



Step 4

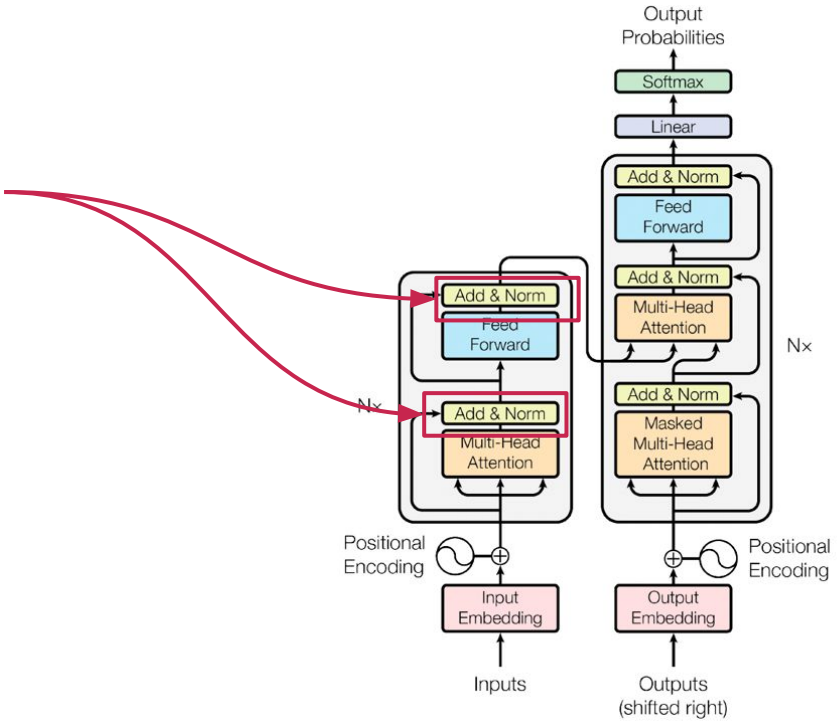
Now we apply a feed-forward network to each token

no interaction here, each token is processed independently

[Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.]

Transformers - Encoder Layer

Add (Residual) & Normalize



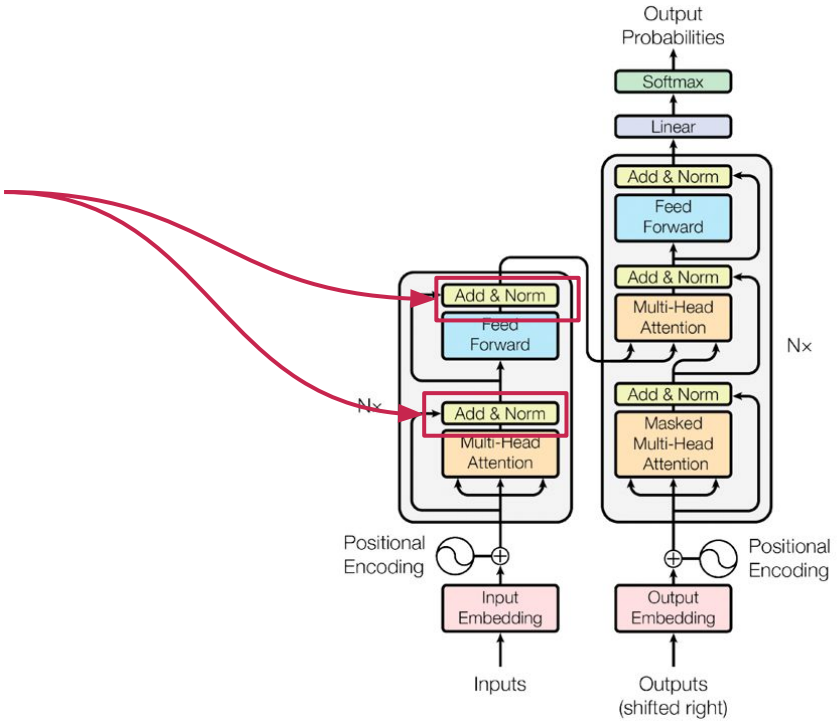
Transformers - Encoder Layer

Add (Residual) & Normalize

Residual Connection:

- Add input back to output
- Helps gradient flow
- Prevents information loss

Output = Input + Layer(x)



Transformers - Encoder Layer

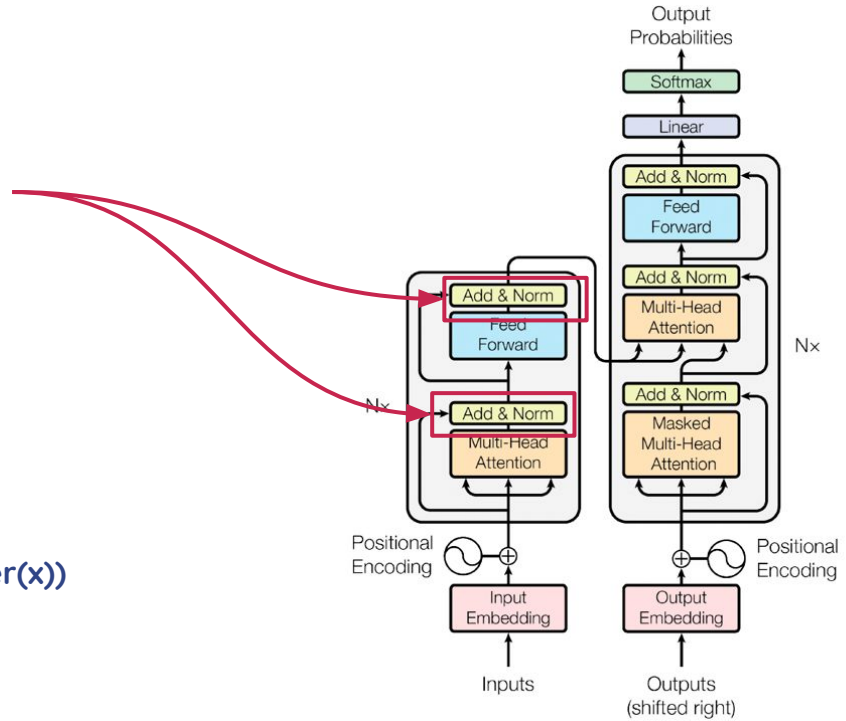
Add (Residual) & Normalize

Residual Connection:

- Add input back to output
- Helps gradient flow
- Prevents information loss

Output = Input + Layer(x)

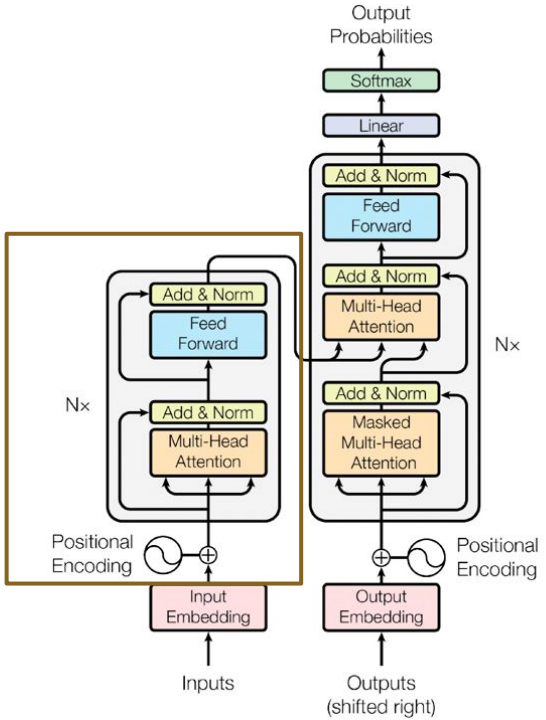
Final Output=LayerNorm(x+Layer(x))



Transformers - Encoder Layer

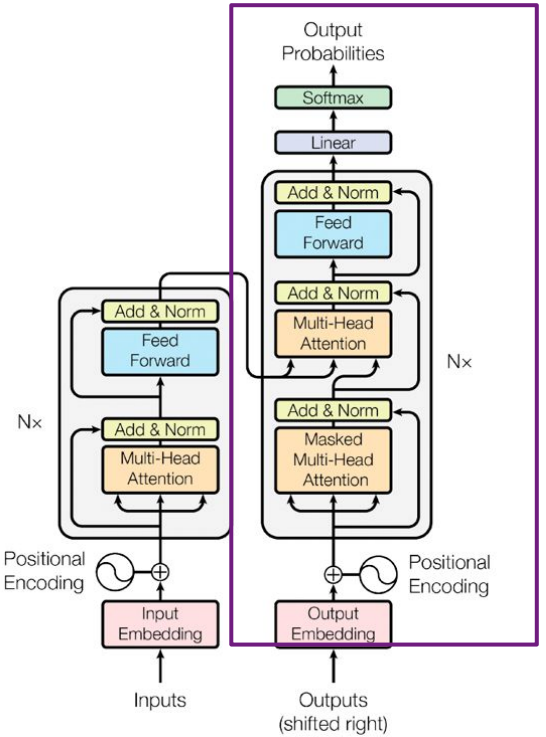
Key takeaways:

- **The encoder extracts meaning from the input sequence.**
- **the encoder creates a rich representation for the decoder to work with.**



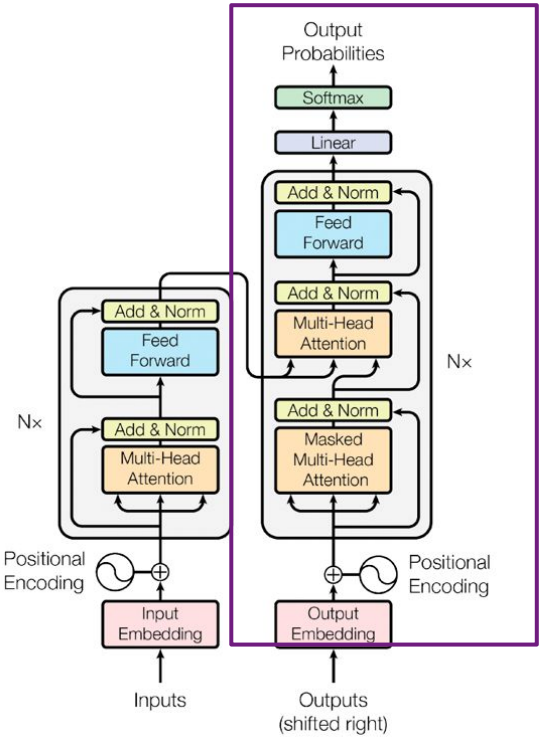
Transformers - Decoder Layer

Any idea how it works ?



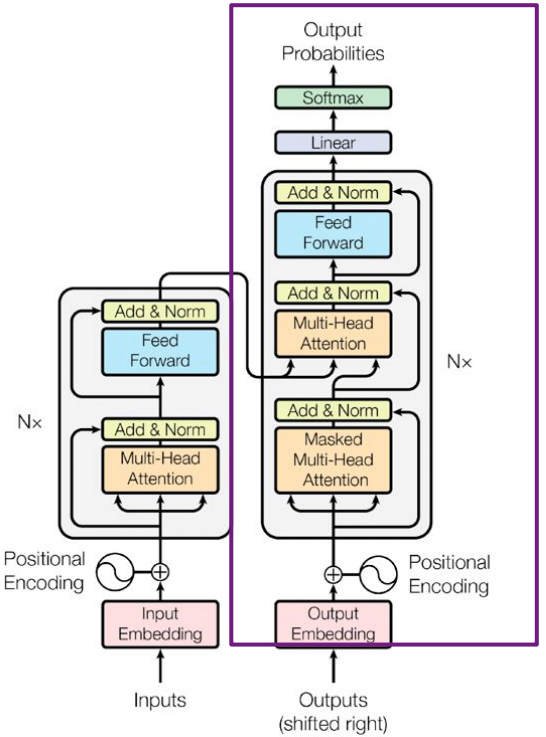
Transformers - Decoder Layer

The decoder is very powerful
and
very complicated



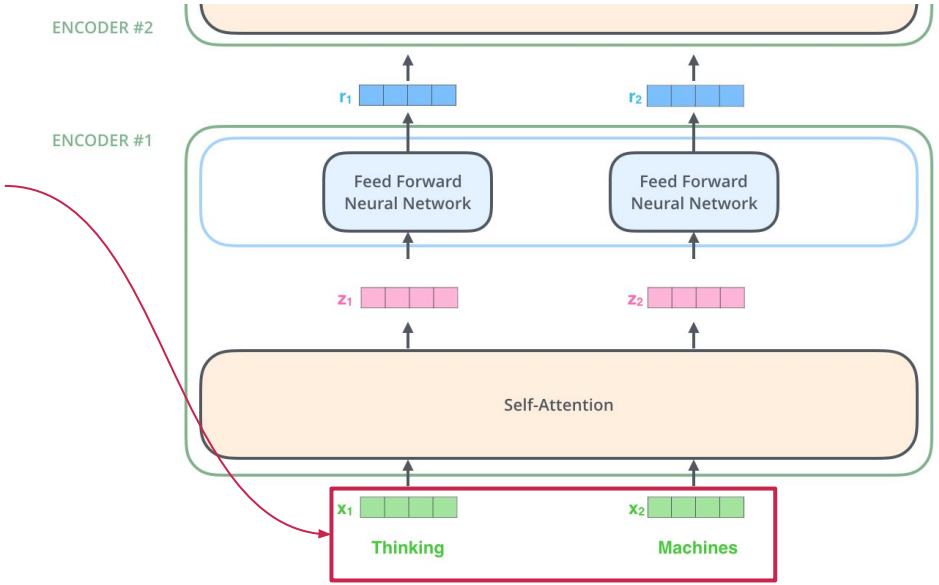
Transformers - Decoder Layer

The decoder is very powerful
and
very complicated
And completely useless for
what we're doing today



Vision Transformer (ViT)

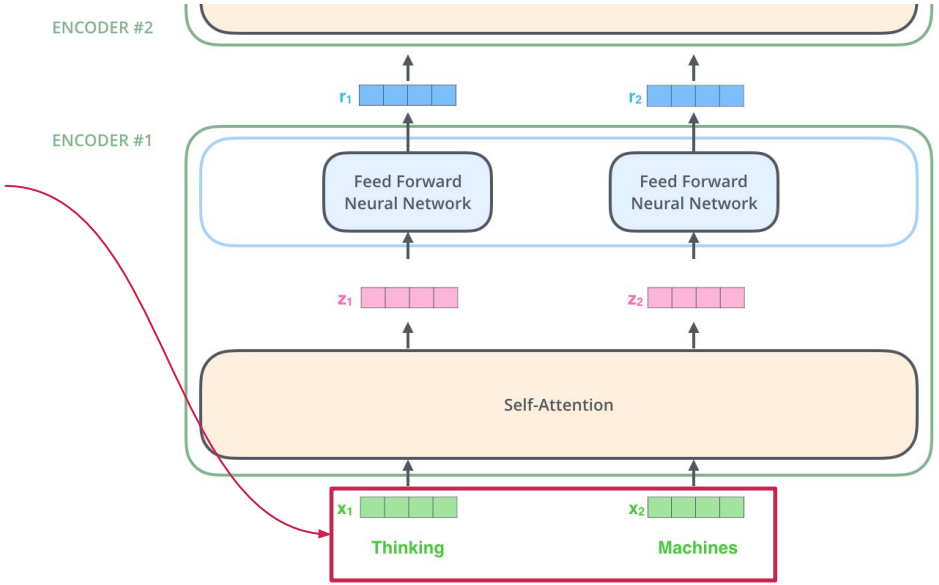
So far, our transformer takes words as input



[Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.]

Vision Transformer (ViT)

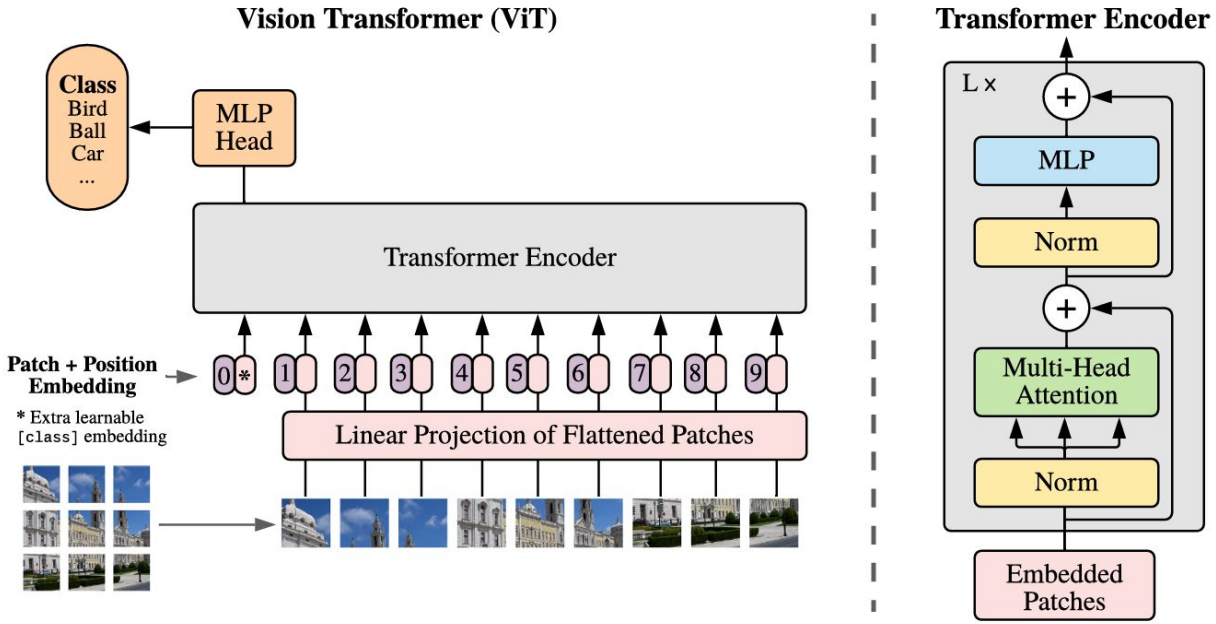
So far, our transformer takes words as input
Now we want to apply this to images…



[Source: The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time.]

Vision Transformer (ViT)

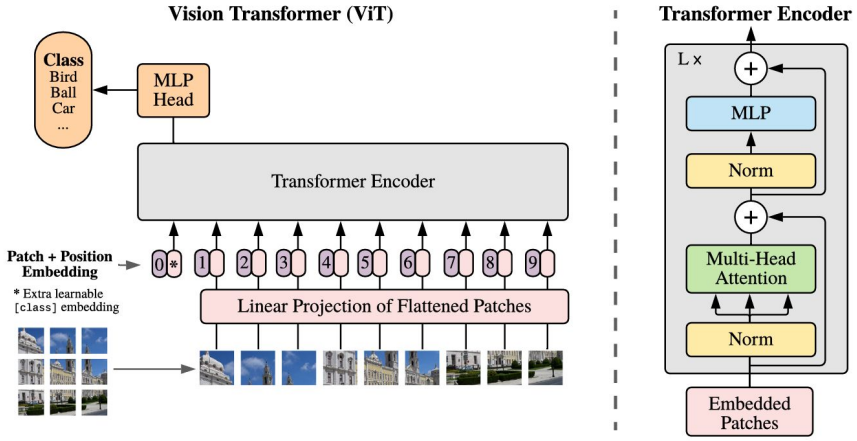
No decoder !!



[Source: An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale]

Vision Transformer (ViT)

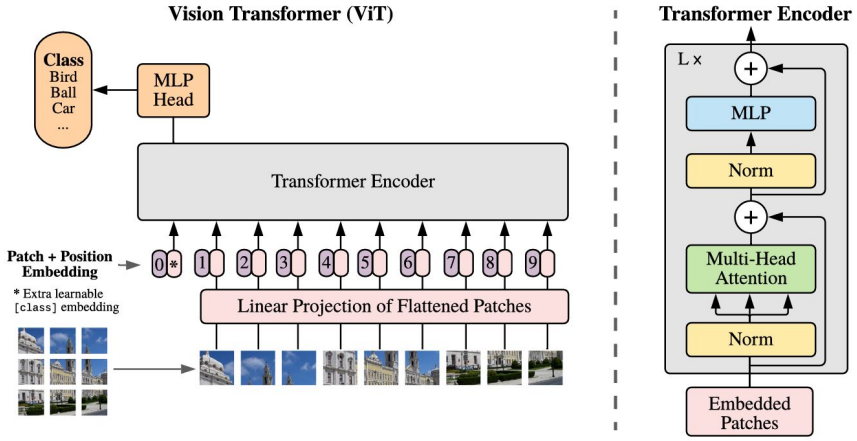
- First full Transformer architecture for Vision
- 16 x 16 patches as ‘words’ : each patch = 16x16x3 (=768d)
- Huge pre-training (Imagenet doesn’t suffice)



[Source: An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale]

Vision Transformer (ViT)

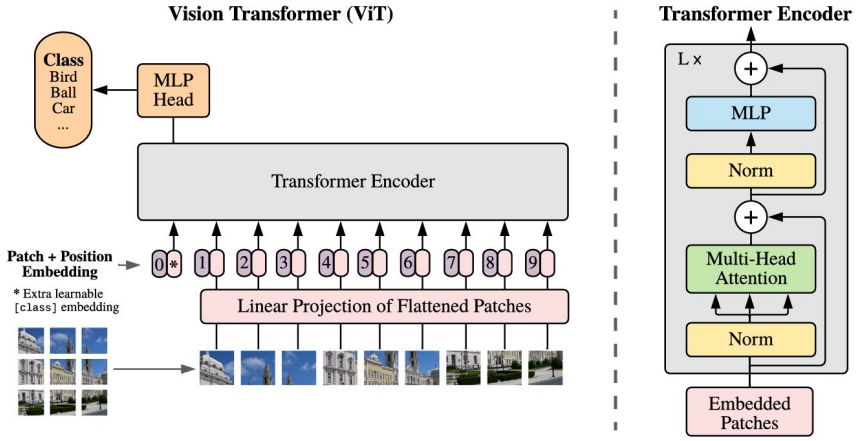
- If ViT is trained on datasets with more than 14M images it can approach or beat state-of-the-art CNNs.



[Source: An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale]

Vision Transformer (ViT)

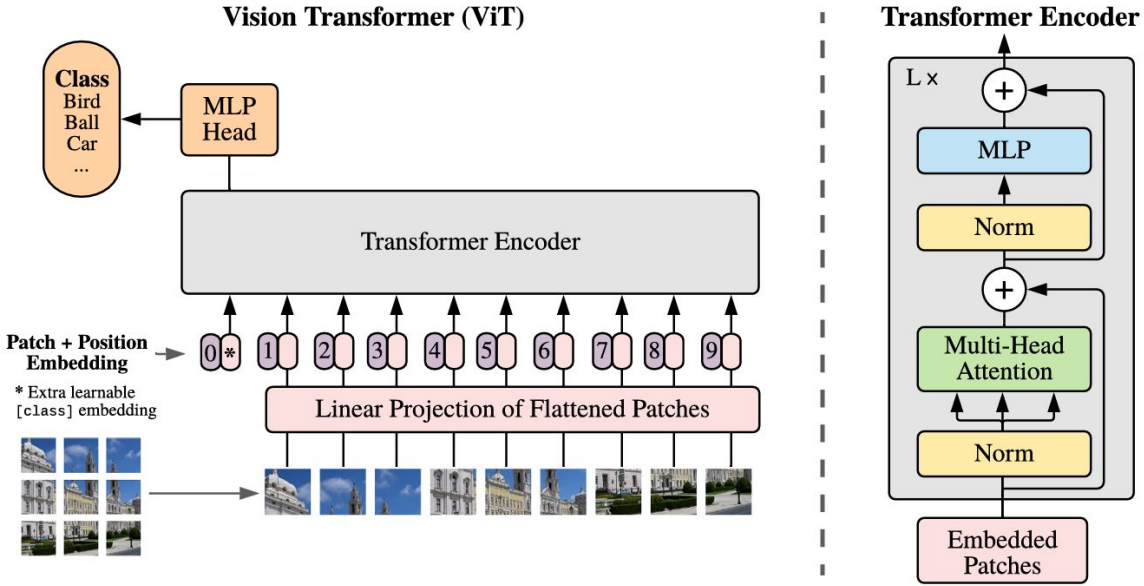
- If ViT is trained on datasets with more than 14M images it can approach or beat state-of-the-art CNNs.
- If not, you better stick with ResNets or EfficientNets .



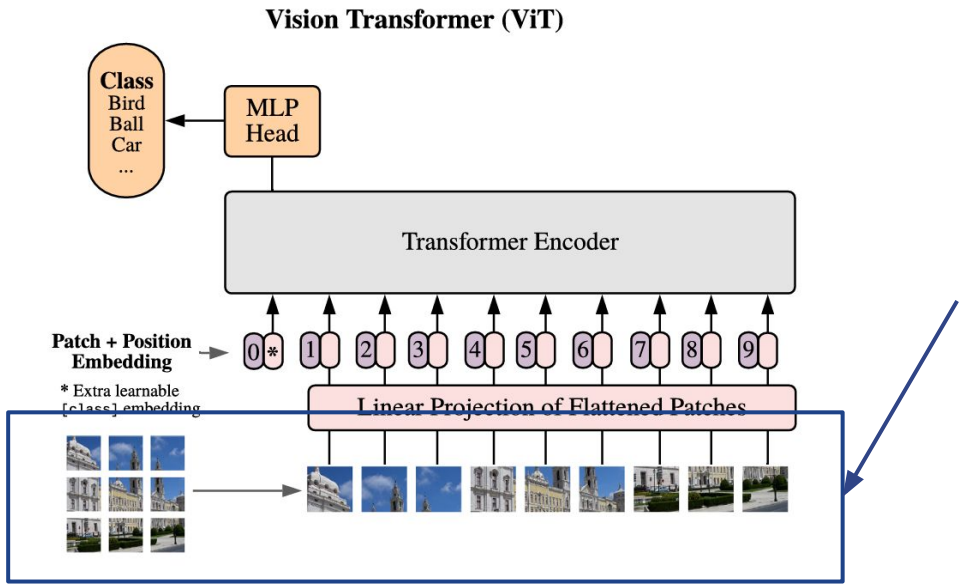
[Source: An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale]

Vision Transformer (ViT)

How does it really works ?



Vision Transformer (ViT)

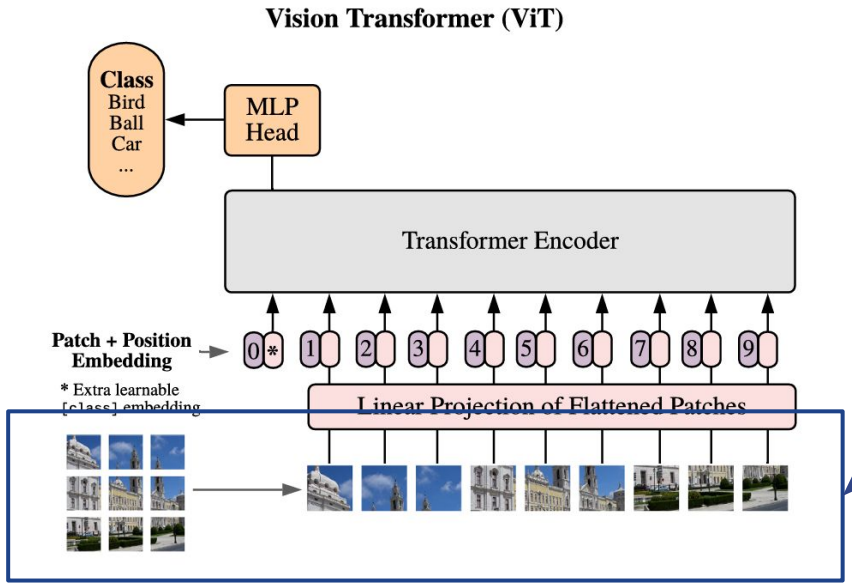


Step 1

Split Image into Patches

- Image is divided into fixed-size patches (e.g., 16×16)
- Each patch becomes a token
- Total patches = sequence length

Vision Transformer (ViT)



Step 1

Split Image into Patches

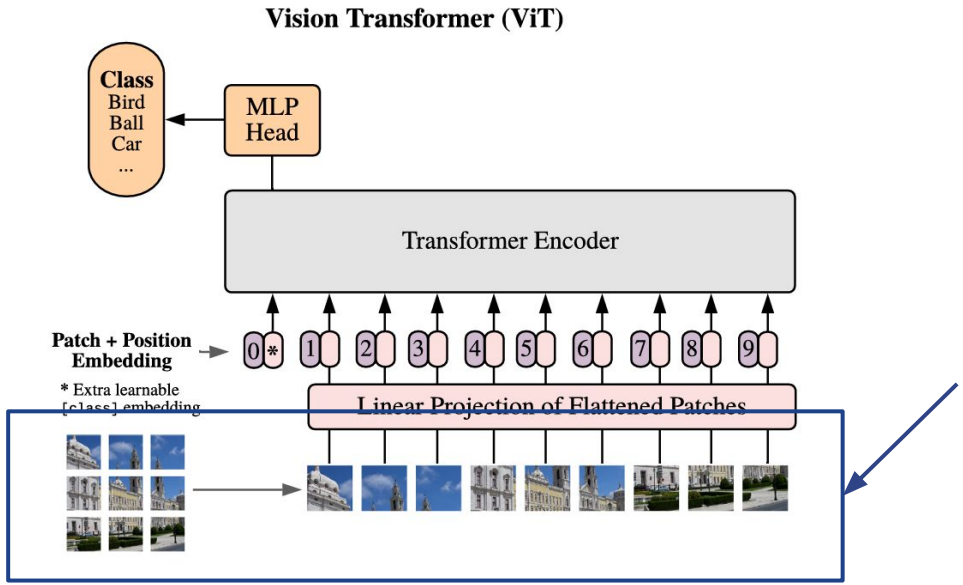
- Image is divided into fixed-size patches (e.g., 16×16)
- Each patch becomes a token
- Total patches = sequence length

Where H , W = image size

P = patch size

$$N = \frac{H}{P} \times \frac{W}{P}$$

Vision Transformer (ViT)

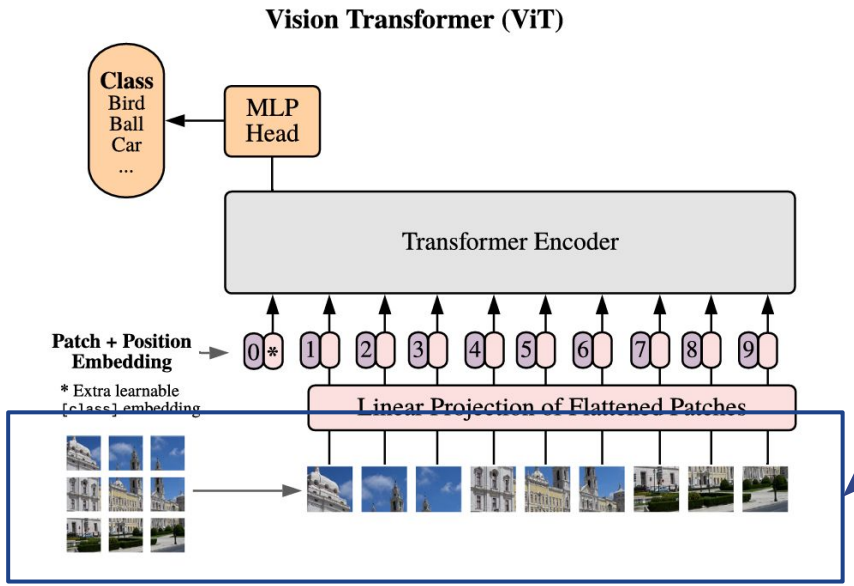


Step 1

Split Image into Patches

Why patches ? why not pixels?

Vision Transformer (ViT)



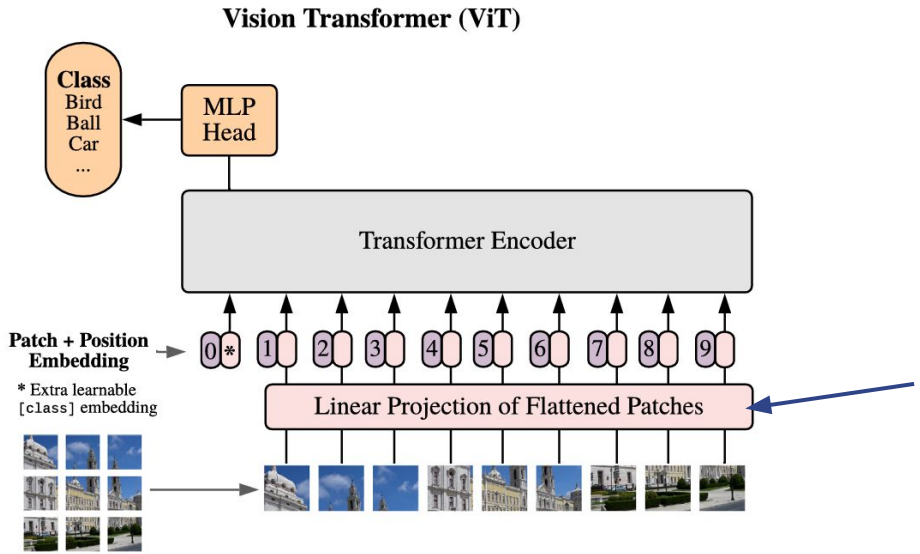
Step 1

Split Image into Patches

Why patches ? why not pixels?

It is relatively easier to understand the relationships between patches of P x P than of a full image Height x Width.

Vision Transformer (ViT)



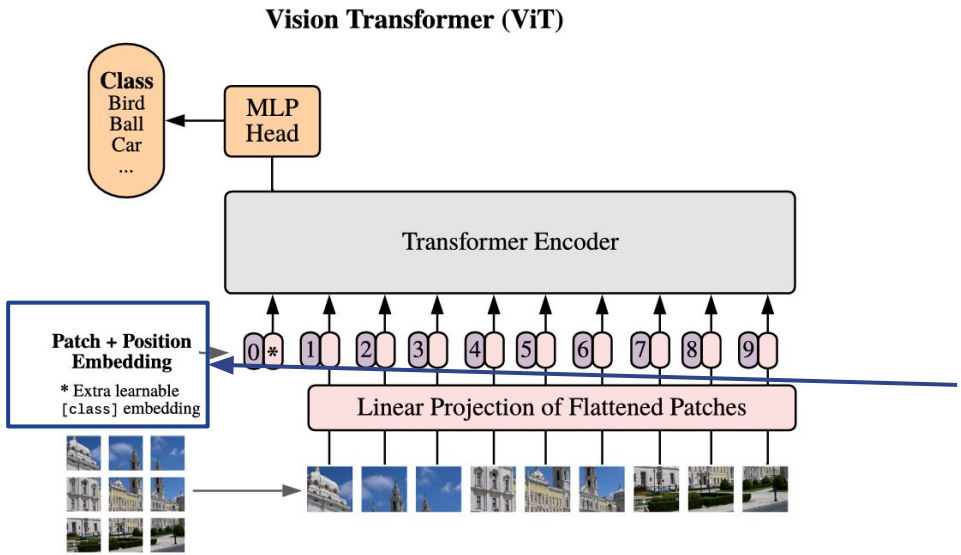
Step 2.1

Patch → Embedding

- Each patch is a small image (e.g., 16×16×3)
- Flatten into a vector
- Apply a linear projection

Each patch → a token vector

Vision Transformer (ViT)

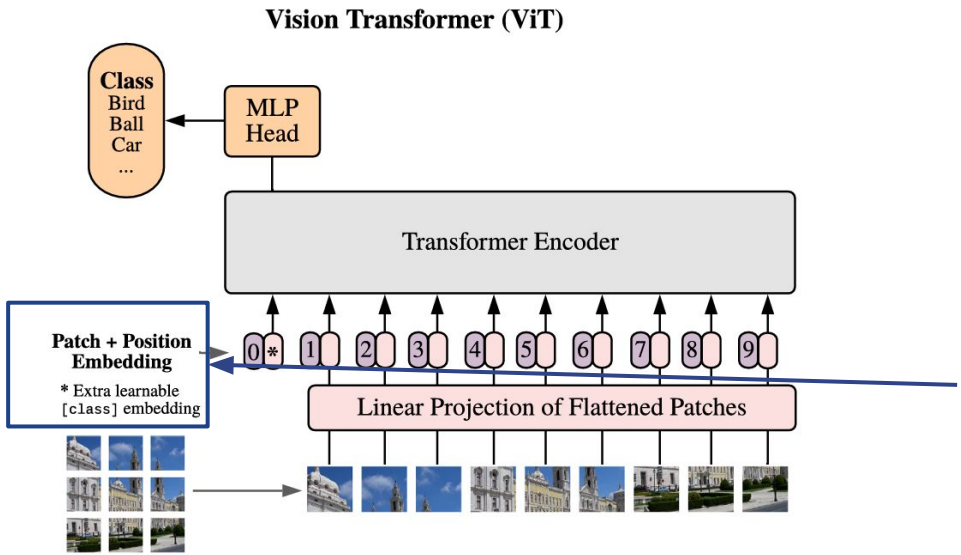


Step 2.2

Positional embeddings

- **Problem** : Transformer does not know spatial order

Vision Transformer (ViT)



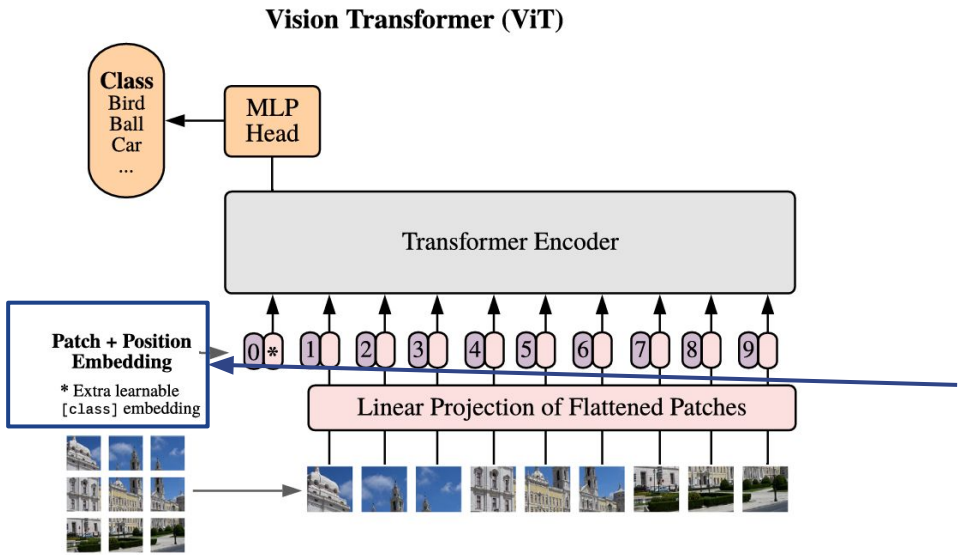
Step 2.2

Positional embeddings

- Problem : Transformer does not know spatial order

- Solution : Add a positional vector to each patch
- Encodes spatial location

Vision Transformer (ViT)



Step 2.2

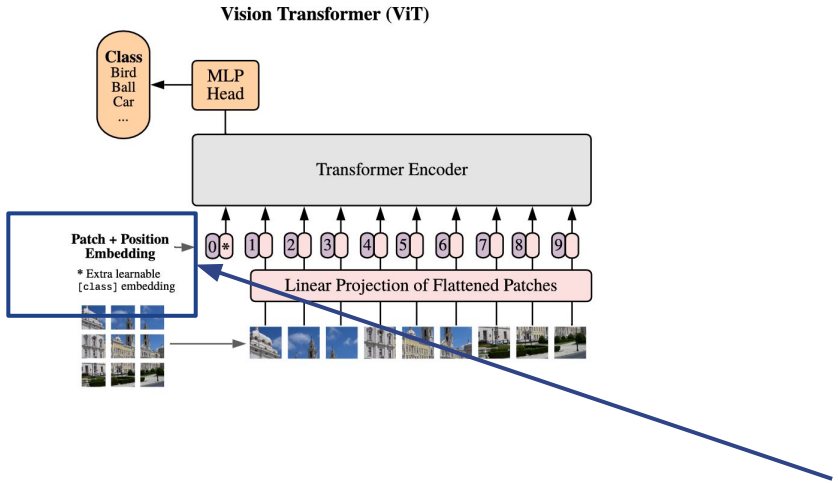
Positional embeddings

- Problem : Transformer does not know spatial order
- Solution : Add a positional vector to each patch
- Encodes spatial location

Embedding = Patch + Position

*** a trainable position**

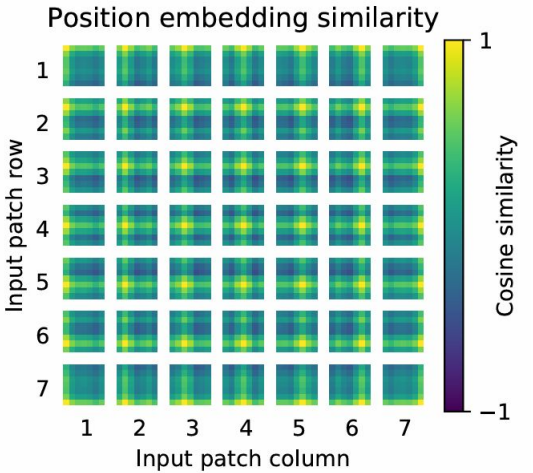
Vision Transformer (ViT)



Similarity between positional embeddings.

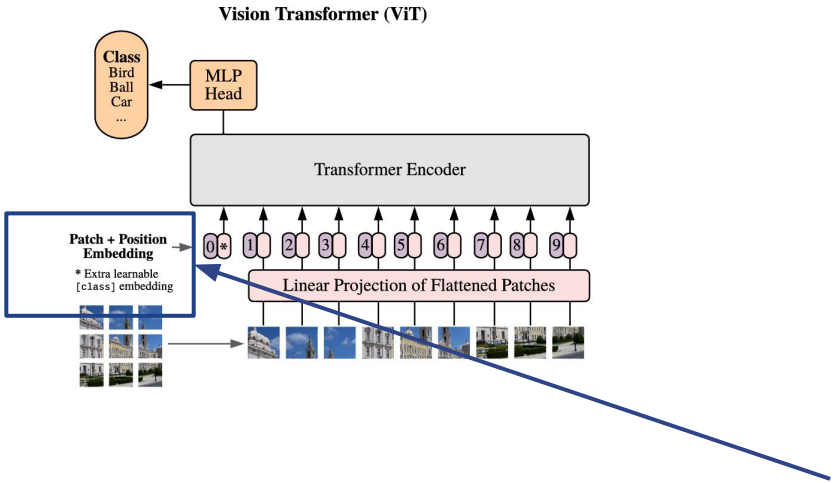
Step 2.2

Positional embeddings



[Source: An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale]

Vision Transformer (ViT)

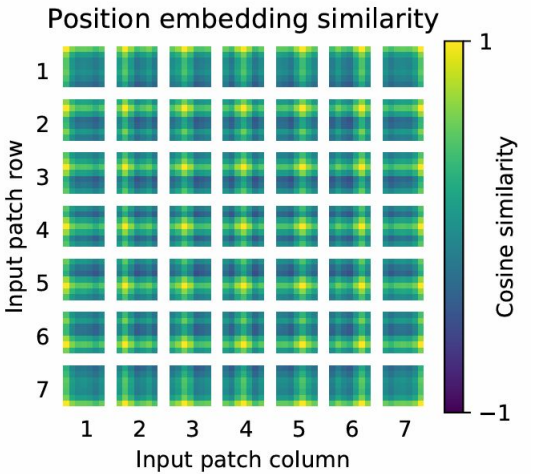


Similarity between positional embeddings.

Why do we have 7x7 heatmap ?

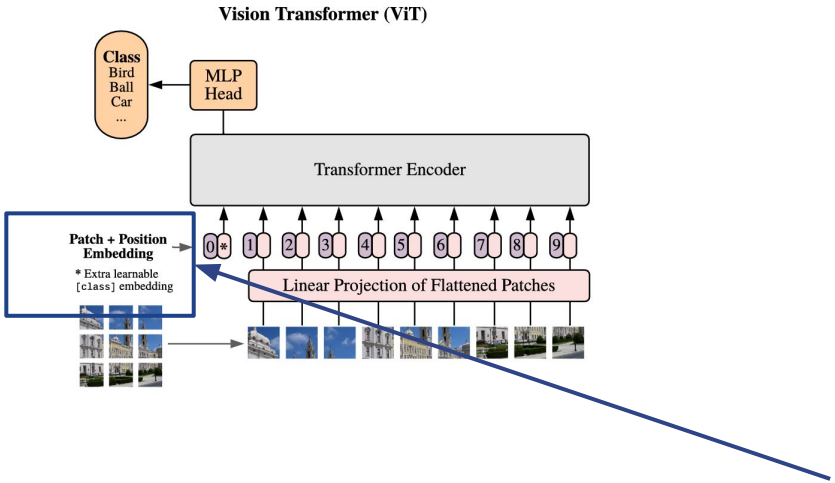
Step 2.2

Positional embeddings



[Source: An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale]

Vision Transformer (ViT)



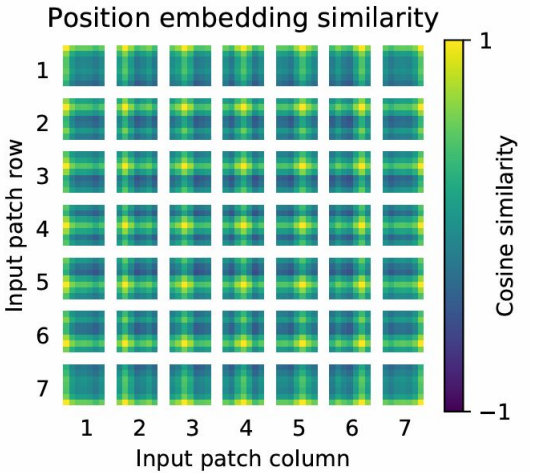
Similarity between positional embeddings.

If we divide a 112×112 image (image size used in the original paper) into 16×16 patches

112/16 = 7x7 patch positions

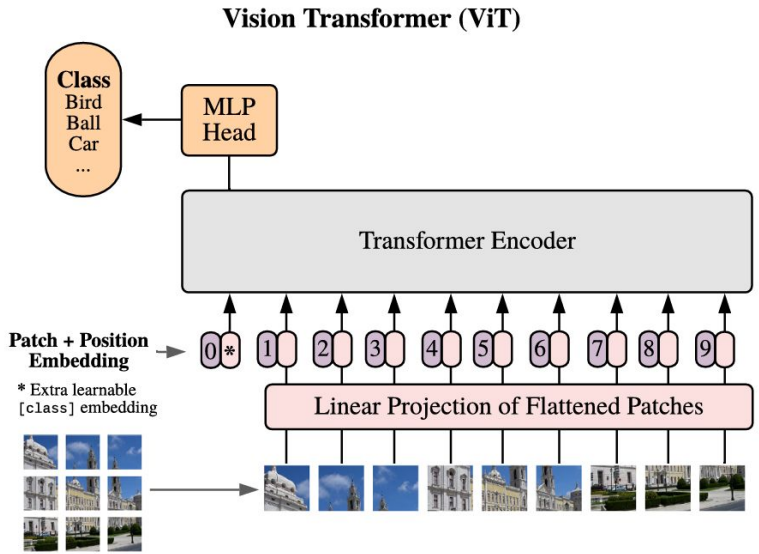
Step 2.2

Positional embeddings



[Source: An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale]

Vision Transformer (ViT)



Step 2.3

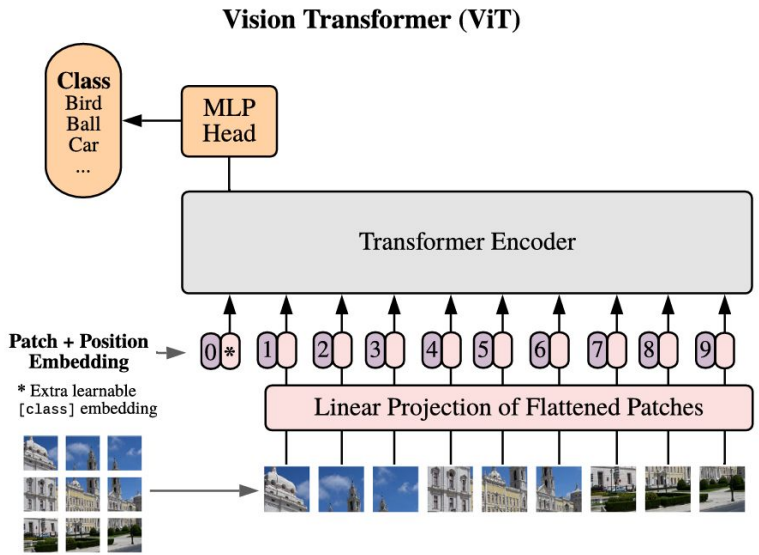
Add [CLS] Token

After splitting the image into patches, converting them into embeddings, and adding positional information

We have sequence of vectors:

x1,x2,x3,...,xn

Vision Transformer (ViT)



Step 2.3

Add [CLS] Token

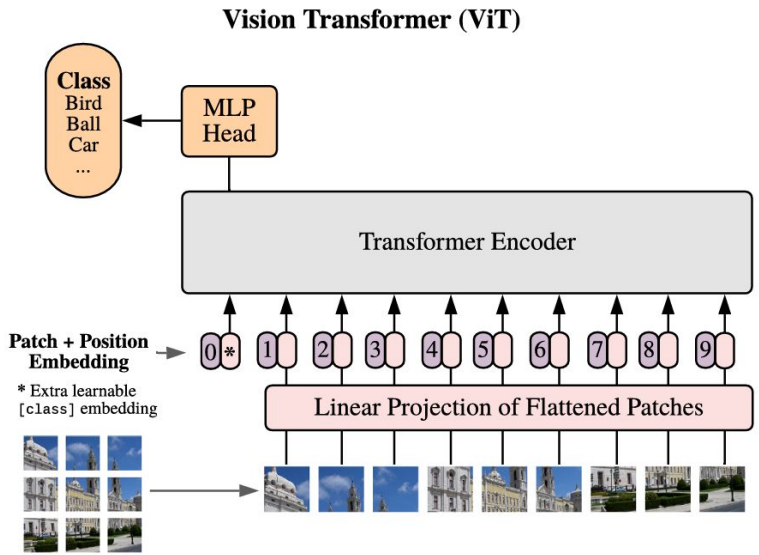
After splitting the image into patches, converting them into embeddings, and adding positional information

We have sequence of vectors:

x1,x2,x3,...,xn

- what the patch contains
- where it is located

Vision Transformer (ViT)



Step 2.3

Add [CLS] Token

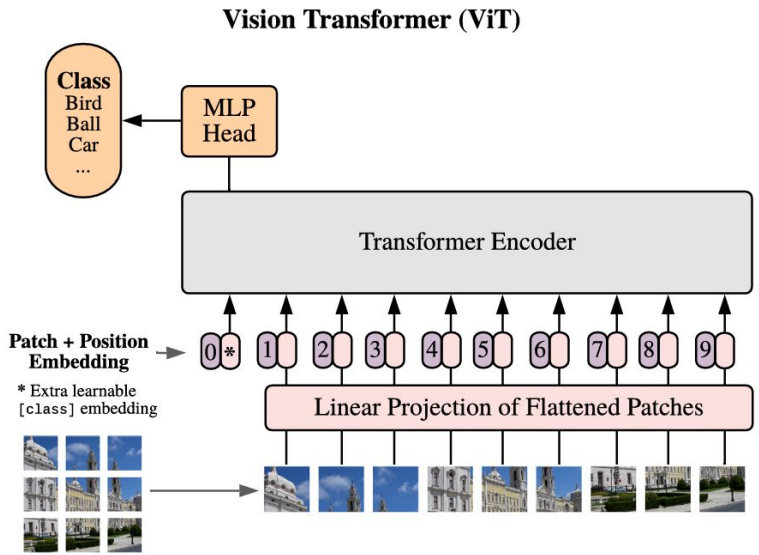
We have sequence of vectors:

x1,x2,x3,...,xn

- what the patch contains
- where it is located

Problem : For a classification task, which vector should we use to make the final prediction

Vision Transformer (ViT)



Step 2.3

Add [CLS] Token

We have sequence of vectors:

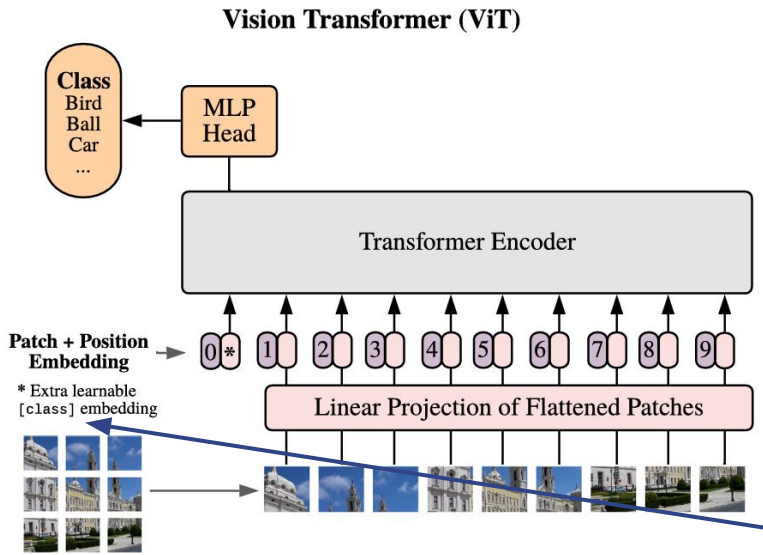
x1,x2,x3,...,xn

- what the patch contains
- where it is located

Problem : For a classification task, which vector should we use to make the final prediction.

Solution : One final representation

Vision Transformer (ViT)



Step 2.3

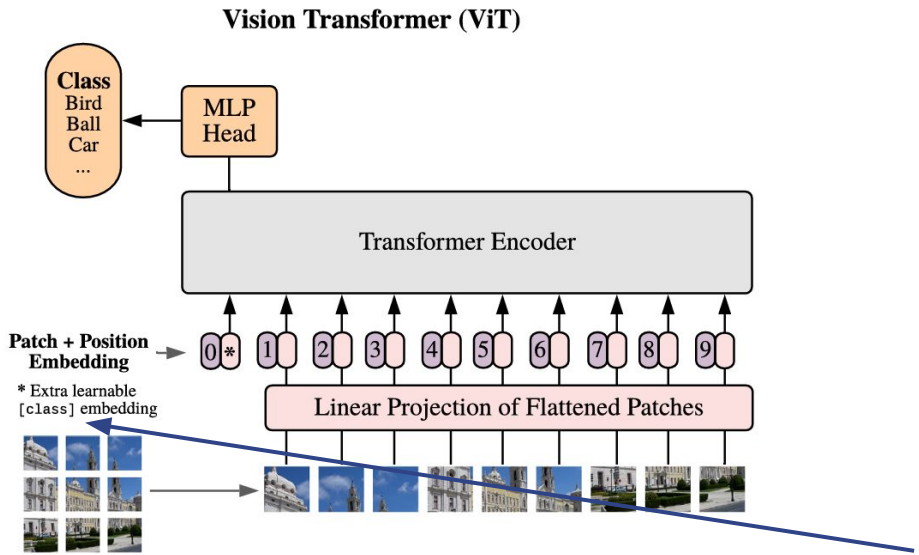
Add [CLS] Token

Problem : For a classification task, which vector should we use to make the final prediction.

Solution : One final representation

We add a special vector called [CLS] at the beginning.

Vision Transformer (ViT)

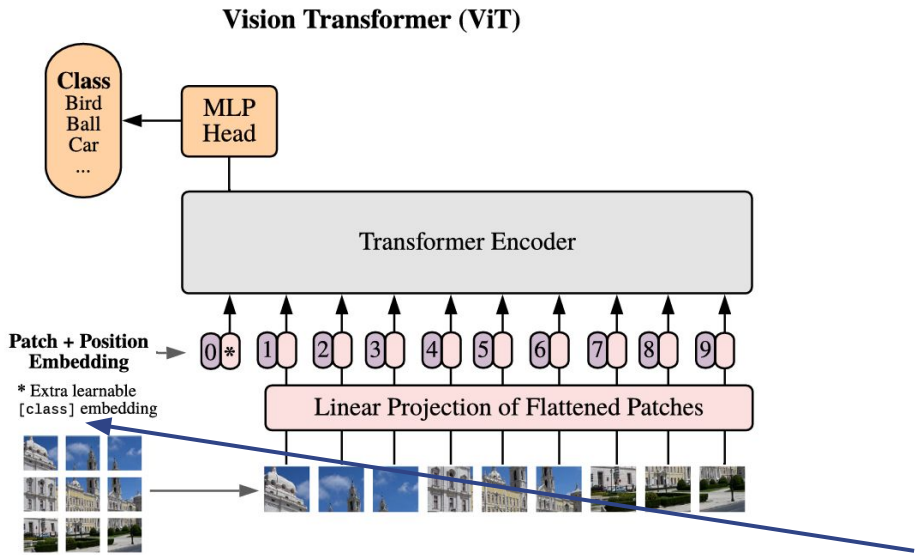


Step 2.3

Add [CLS] Token

- We add a special vector called [CLS] at the beginning.
- At the beginning, [CLS] is just a random vector, but during attention, it interacts with all patches

Vision Transformer (ViT)



Step 2.3

Add [CLS] Token

- We add a special vector called [CLS] at the beginning.
- At the beginning, [CLS] is just a random vector, but during attention, it interacts with all patches
- The [CLS] token becomes a summary of the whole image (close to CNN pooling)

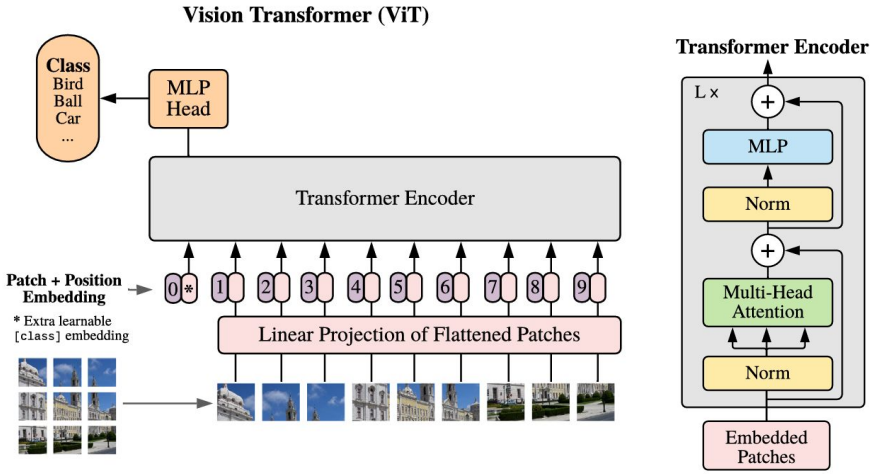
Vision Transformer (ViT)

Same transformer encoder as in NLP

Step 3

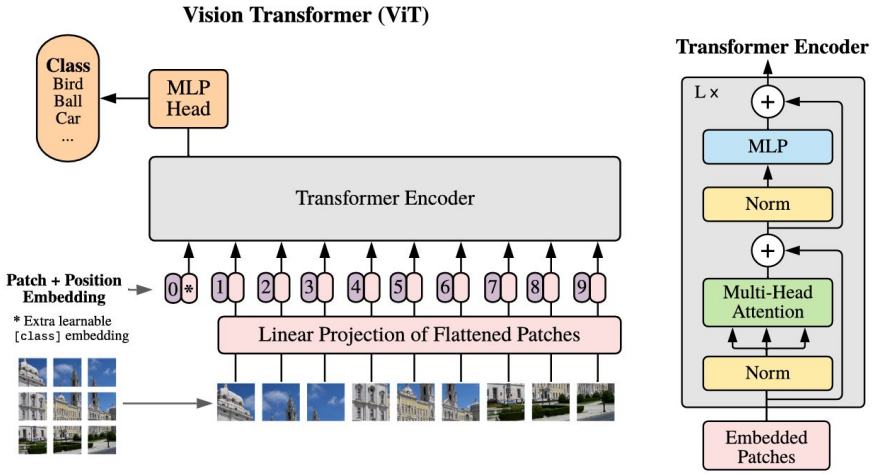
Transformer Encoder

What we have : [CLS],x1,x2,...,xn



Vision Transformer (ViT)

Same transformer encoder as in NLP



Step 3

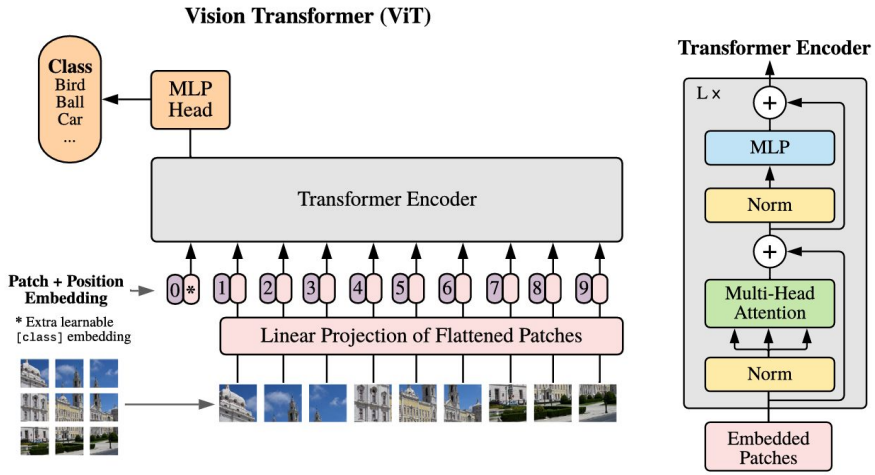
Transformer Encoder

What we have : [CLS],x1,x2,...,xn

We pass it through multiple encoder layers

Vision Transformer (ViT)

Same transformer encoder as in NLP



Step 3

Transformer Encoder

What we have : [CLS],x1,x2,...,xn

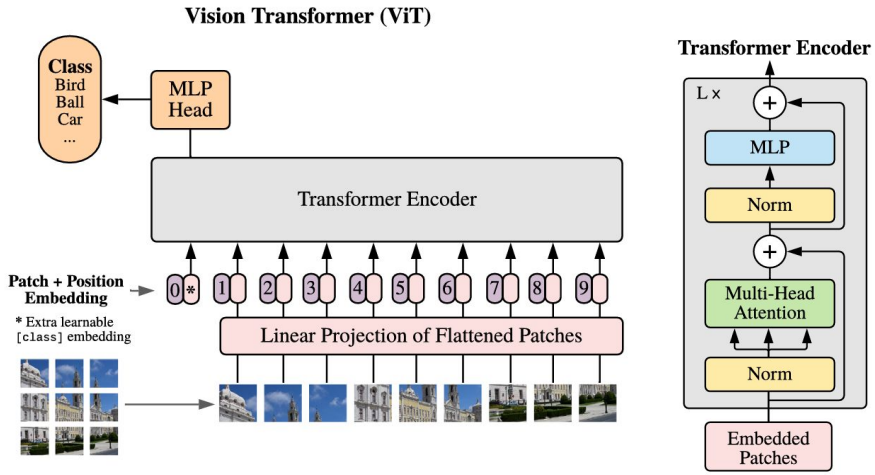
We pass it through multiple encoder layers

Each layer applies:

- Self-attention
- Feed-forward

Vision Transformer (ViT)

Same transformer encoder as in NLP



Step 3

Transformer Encoder

What we have : [CLS],x1,x2,...,xn

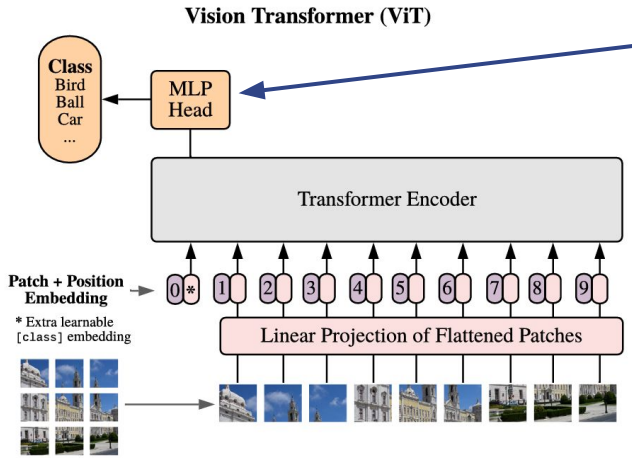
We pass it through multiple encoder layers

Each layer applies:

- Self-attention
- Feed-forward

The CLS gradually becomes a global representation of the image

Vision Transformer (ViT)



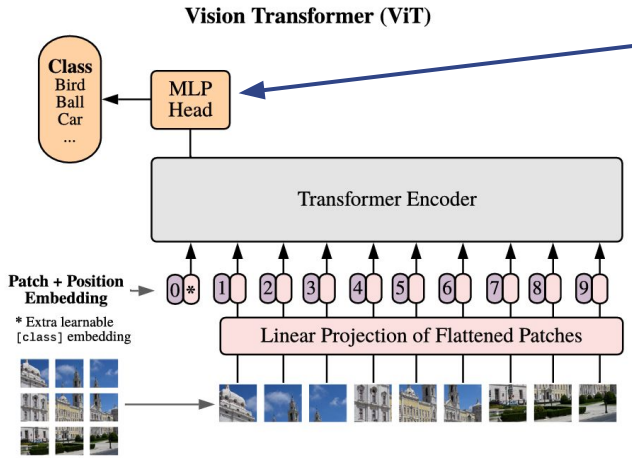
Step 4

Image Classification

What we have : a global representation :

[CLS]

Vision Transformer (ViT)



Step 4

Image Classification

What we have : a global representation : [CLS]

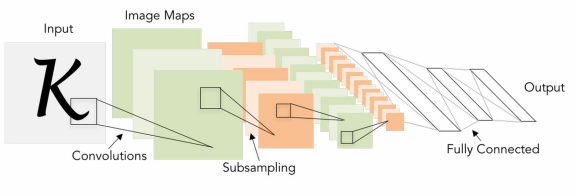
- Pass it through a small **multi-layer perceptron** (MLP)
- Output class probabilities

ViT in a nutshell

[Source: Gif Link]

CNN vs Vision Transformer

- Local receptive field
- Processes entire images directly
- Works well with small data



- Attention
- Divides images into patches and processes them as sequences
- Needs large data

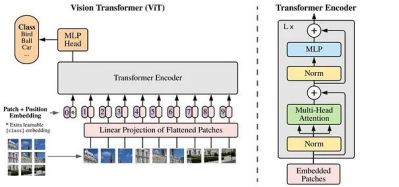


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

Evolution of Vision Transformers

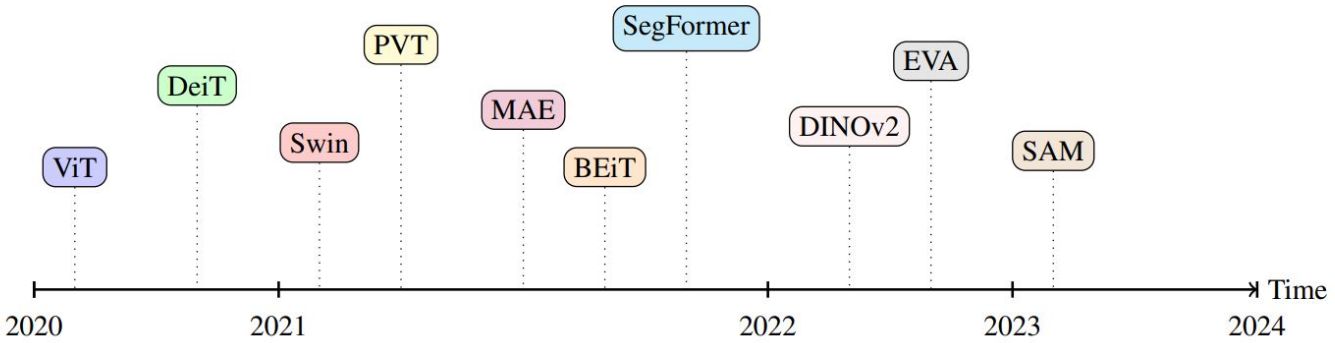


Figure 1. Evolution timeline of Vision Transformer architectures showing major milestones from 2020 to 2024. Each architecture introduced key innovations that advanced the field.

[Source: Laurier, Linda. (2025). Vision Transformers: From Patch-Based Processing to Visual Intelligence.]

Evolution of Vision Transformers

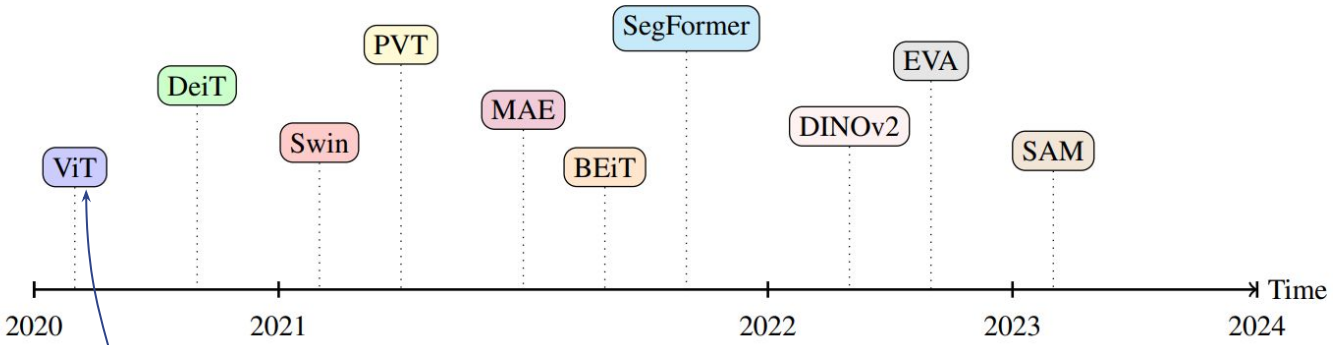


Figure 1. Evolution timeline of Vision Transformer architectures showing major milestones from 2020 to 2024. Each architecture introduced key innovations that advanced the field.

Need huge datasets

[Source: Laurier, Linda. (2025). Vision Transformers: From Patch-Based Processing to Visual Intelligence.]

Evolution of Vision Transformers

How to train transformers with less data using distillation

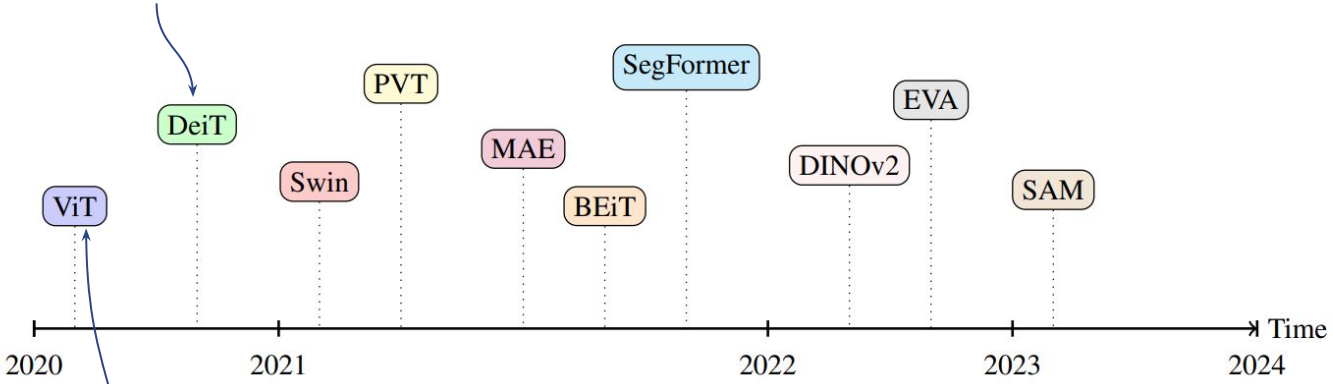


Figure 1. Evolution timeline of Vision Transformer architectures showing major milestones from 2020 to 2024. Each architecture introduced key innovations that advanced the field.

Need huge datasets

[Source: Laurier, Linda. (2025). Vision Transformers: From Patch-Based Processing to Visual Intelligence.]

Evolution of Vision Transformers

How to train transformers with less data using distillation

Hierarchical Transformers

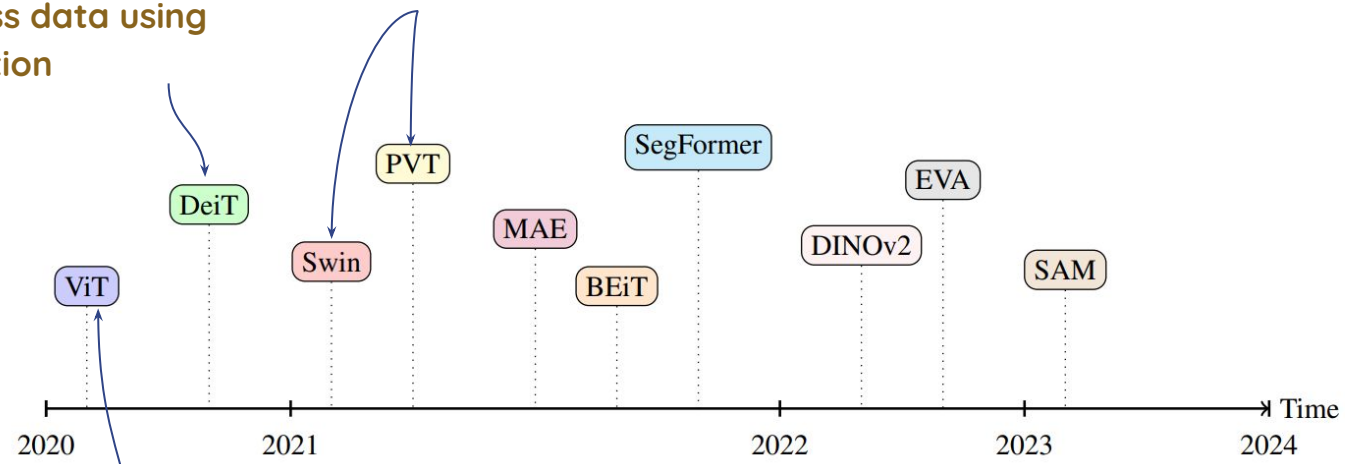


Figure 1. Evolution timeline of Vision Transformer architectures showing major milestones from 2020 to 2024. Each architecture introduced key innovations that advanced the field.

Need huge datasets

[Source: Laurier, Linda. (2025). Vision Transformers: From Patch-Based Processing to Visual Intelligence.]

Evolution of Vision Transformers

How to train transformers with less data using distillation

Hierarchical Transformers

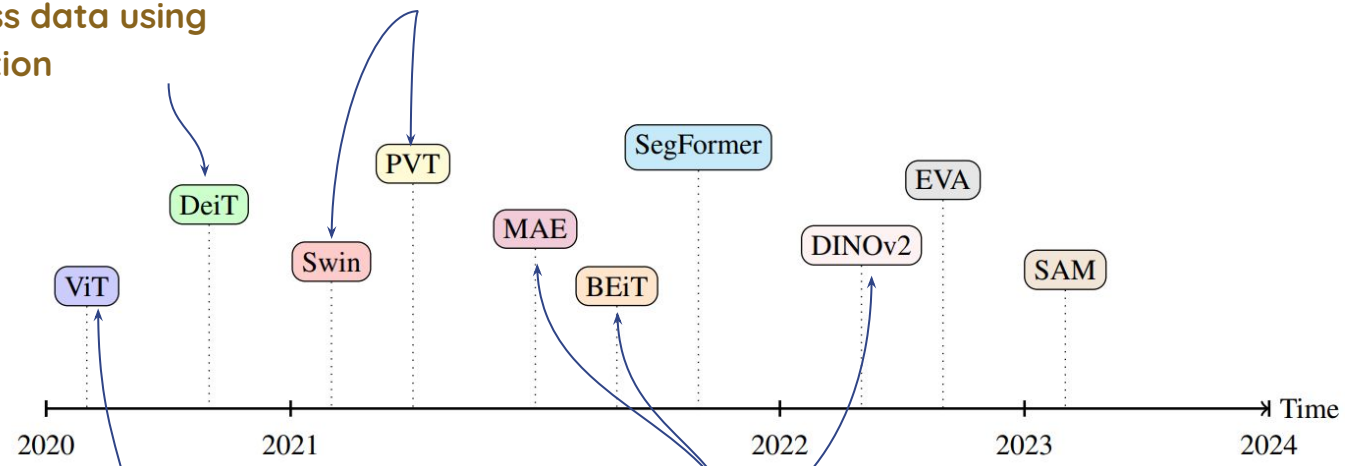


Figure 1. Evolution timeline of Vision Transformer architectures showing major milestones from 2020 to 2024. Each architecture introduced key innovations that advanced the field.

Need huge datasets

Self-Supervised Learning

[Source: Laurier, Linda. (2025). Vision Transformers: From Patch-Based Processing to Visual Intelligence.]

Evolution of Vision Transformers

How to train transformers with less data using distillation

Hierarchical Transformers

Transformers moved beyond classification

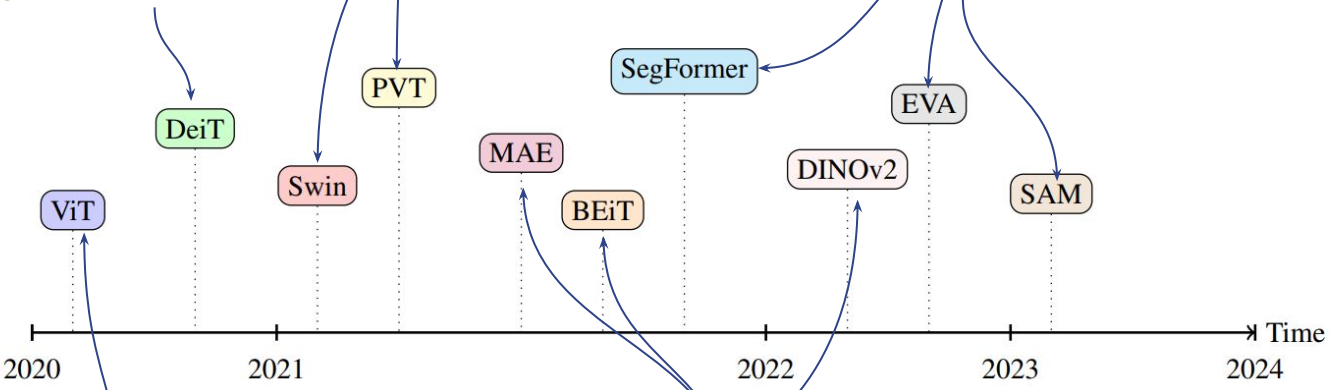


Figure 1. Evolution timeline of Vision Transformer architectures showing major milestones from 2020 to 2024. Each architecture introduced key innovations that advanced the field.

Need huge datasets

Self-Supervised Learning

[Source: Laurier, Linda. (2025). Vision Transformers: From Patch-Based Processing to Visual Intelligence.]

Object Detection

References I

Books

• Computer Vision: Models, Learning, and Inference (2012). https://amzn.to/2rxrdOF
• Programming Computer Vision with Python (2012). https://amzn.to/2QKTaAL
• Multiple View Geometry in Computer Vision (2004). https://amzn.to/2LfHLE8
• Computer Vision: Algorithms and Applications (2010). https://amzn.to/2LcIt4J
• Computer Vision: A Modern Approach (2002). https://amzn.to/2rv7AqJ
• Deep Learning for Computer Vision : Image Classification, Object Detection and Face Recognition in Python(2019). https://machinelearningmastery.com/deep-learning-for-computer-vision/
• Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow — Aurélien Géron (2019) https://amzn.to/3uZbN6Z
• Hands-On Large Language Models - Jay Alammar & Grootendorst

Articles & Online Resources

• Krizhevsky, Sutskever, Hinton (2012) — ImageNet Classification with Deep CNNs (AlexNet — deep learning breakthrough)
• Improved Texture Networks: Maximizing Quality and Diversity in Feed-forward Stylization and Texture Synthesis, 2017
• "Computer vision," Wikipedia. https://en.wikipedia.org/wiki/Computer_vision
• "Machine vision," Wikipedia. https://en.wikipedia.org/wiki/Machine_vision
• "Digital image processing," Wikipedia. https://en.wikipedia.org/wiki/Digital_image_processing

References II

• “Computer vision,” Wikipedia. https://en.wikipedia.org/wiki/Computer_vision
• https://medium.com/data-reply-it-datatech
• https://huggingface.co/blog/RDTvlokip/quand-l-ia-apprend-de-l-experience-comme-toi
• Zhuang et al., “A Comprehensive Survey on Transfer Learning,” IEEE, 2020.
• https://jalammar.github.io/illustrated-transformer/
• https://sh-tsang.medium.com/review-vision-transformer-vit-406568603de0
•Laurier, Linda. (2025). Vision Transformers: From Patch-Based Processing to Visual Intelligence.
•Training Convolutional Neural Networks: What Is Machine Learning” Analog Devices. https://www.analog.com/en/resources/analog-dialogue/articles/training-convolutional-neural-networks-what-is-machine-learning-part-2.html