Neural Networks from Scratch

Transformers

Caleb Hallinan

01/13/2026

10x Xenium Breast Cancer Data

Announcements

- Lecture and code from yesterday is on GitHub
- Was not able to get classroom :/
- Today will go over Transformers, code on GitHub
- I will also share project details for you to get started on!

A few questions from reflection cards

How does back propagation work through max pooling?

- MaxPool is like a “winner-take-all” selection:
  - forward: pick the winner (the max)
  - backward: only the winner is responsible for the output, so only the winner receives the gradient

Input

$$X = \begin{bmatrix} 1 & 3 & 2 & 0 \\ 4 & 6 & 1 & 2 \\ 5 & 2 & 7 & 3 \\ 0 & 1 & 4 & 8 \end{bmatrix}$$

output

$$Y = \begin{bmatrix} 6 & 2 \\ 5 & 8 \end{bmatrix}$$

Gradient from next layer

$$\frac{\partial L}{\partial Y} = \begin{bmatrix} 10 & 20 \\ 30 & 40 \end{bmatrix}$$

Then this is backprop

$$\frac{\partial L}{\partial X} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 10 & 0 & 20 \\ 30 & 0 & 0 & 0 \\ 0 & 0 & 0 & 40 \end{bmatrix}$$

If tied, typically first max index is used

Can a CNN do more with input images besides classification?

- For sure! This is my research model – CNN for predicting continuous variables (gene expression)

```
GeneExpressionPredictor(
  (feature_extractor): ResNet(
    (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
    (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (relu): ReLU()
    (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
    (layer1): Sequential(
      (0): Bottleneck(
        (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu_1): ReLU()
        (relu_2): ReLU()
        (relu_3): ReLU()
        (downsample): Sequential(
          (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

⋮

CNN part

```
(feature_layers): Sequential(
  (0): Linear(in_features=2048, out_features=2048, bias=True)
  (1): BatchNorm1d(2048, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (2): ReLU()
  (3): Dropout(p=0.2, inplace=False)
  (4): Linear(in_features=2048, out_features=1024, bias=True)
  (5): BatchNorm1d(1024, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (6): ReLU()
  (7): Dropout(p=0.2, inplace=False)
  (8): Linear(in_features=1024, out_features=512, bias=True)
  (9): BatchNorm1d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (10): ReLU()
  (11): Dropout(p=0.2, inplace=False)
  (12): Linear(in_features=512, out_features=256, bias=True)
  (13): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (14): ReLU()
  (15): Dropout(p=0.2, inplace=False)
)
(output): Linear(in_features=256, out_features=306, bias=True)
```

Parameters we have discussed!

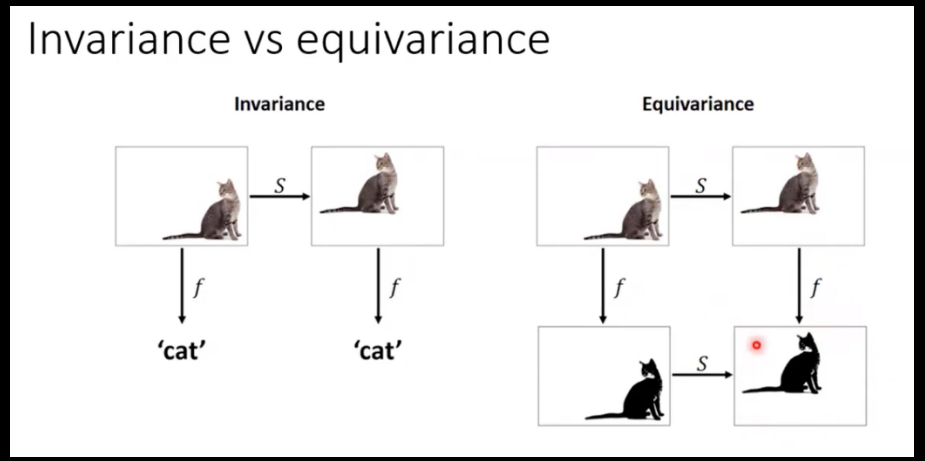
Project Details

Transformers

CNNs Background

- For nearly a decade, Convolutional Neural Networks (CNNs) dominated computer vision tasks like image classification, segmentation, and object detection
  - AlexNet (2012), VGG, ResNet, EfficientNet

- However, CNNs are built with inductive biases:
  - Locality: nearby pixels are more related
  - Translation equivariance: patterns detected in one location can generalize to others



- Equivariance helps reuse the same detector everywhere (why conv works).
- Invariance helps generalization: the model focuses on what is present, not where.

Transformer Revolution

- Transformers (Vaswani et al., 2017) revolutionized natural language processing (e.g., BERT, GPT)
  - Key advantage: self-attention enables global context modeling

- Unlike RNNs or CNNs, transformers can model long-range dependencies efficiently

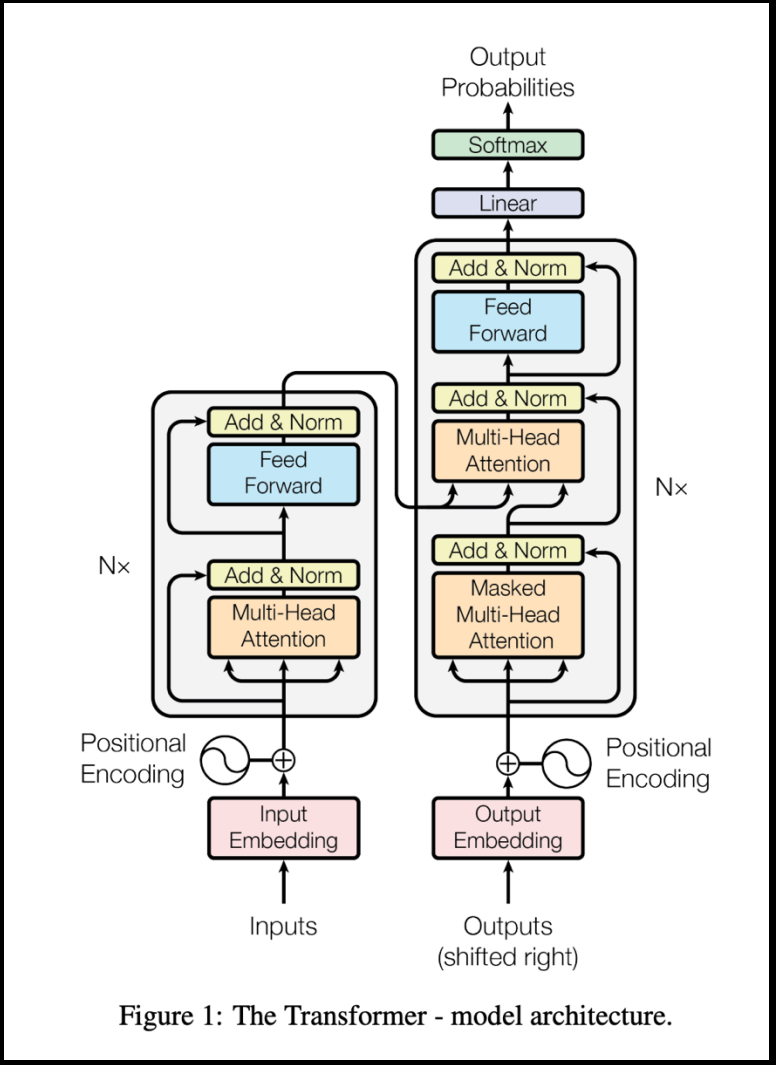


Figure 1: The Transformer - model architecture.

Attention is all you need [PDF] neurips.cc

A Vaswani, N Shazeer, N Parmar... - Advances in neural ..., 2017 - proceedings.neurips.cc

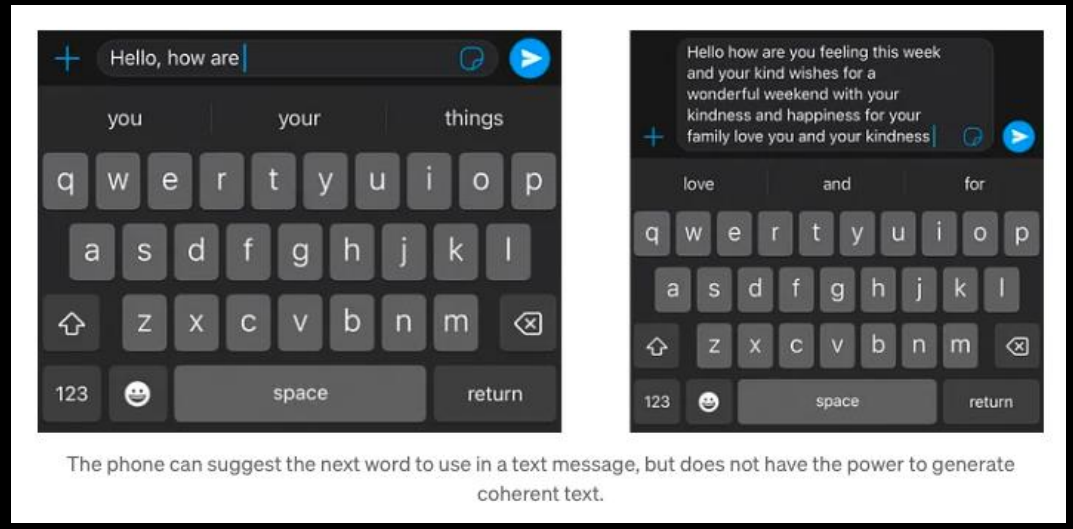
... to attend to **all** positions in the decoder up to and including that position. **We need** to prevent ... **We** implement this inside of scaled dot-product **attention** by masking out (setting to $-\infty$) ...

☆ Save 99 Cite Cited by 211635 Related articles All 70 versions Import into BibTeX »

https://arxiv.org/pdf/1706.03762

What do they do?

- “Imagine that you’re writing a text message on your phone. After each word, you may get three words suggested to you. For example, if you type “Hello, how are”, the phone may suggest words such as “you”, or “your” as the next word.”



The phone can suggest the next word to use in a text message, but does not have the power to generate coherent text.

Command: Write a story.
Response: Once

Next command: Write a story. Once
Response: upon

Next command: Write a story. Once upon
Response: a

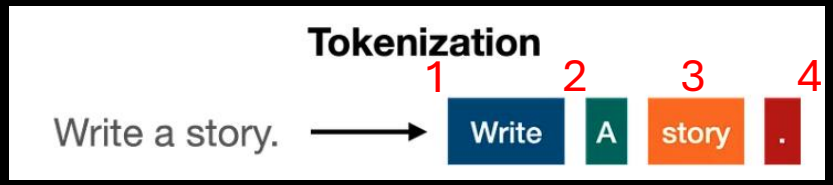
Next command: Write a story. Once upon a
Response: time

Next command: Write a story. Once upon a time
Response: there

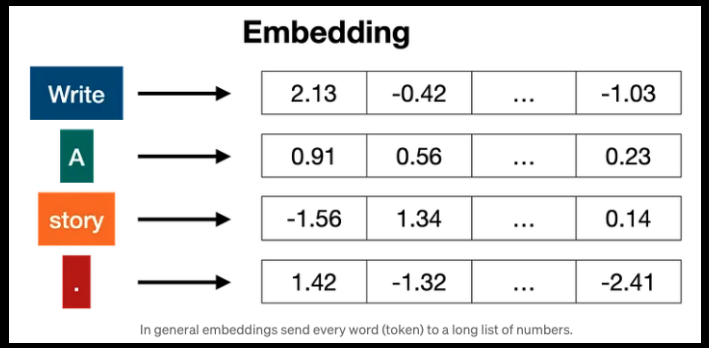
https://medium.com/@amanatulla1606/transformer-architecture-explained-2c49e2257b4c

Transformer Embedded Patches

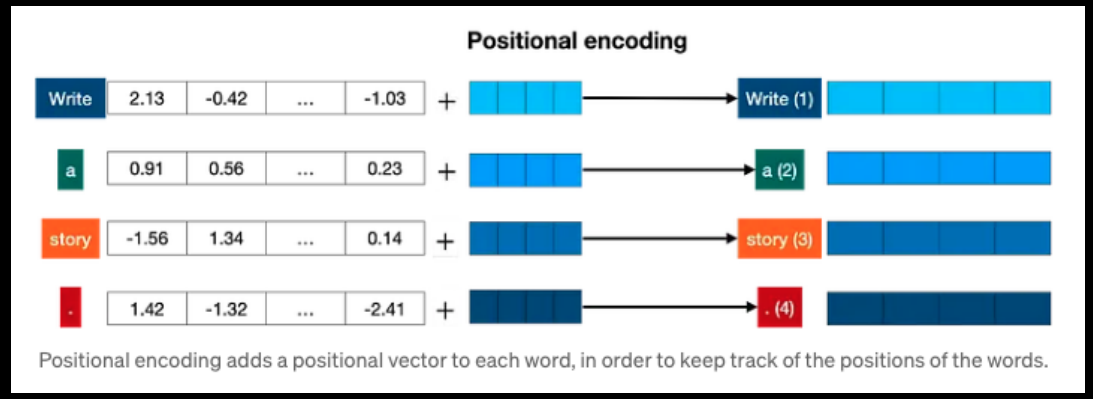
- Tokenization
  - Turns raw text into a sequence of integer IDs that the model can embed and process.

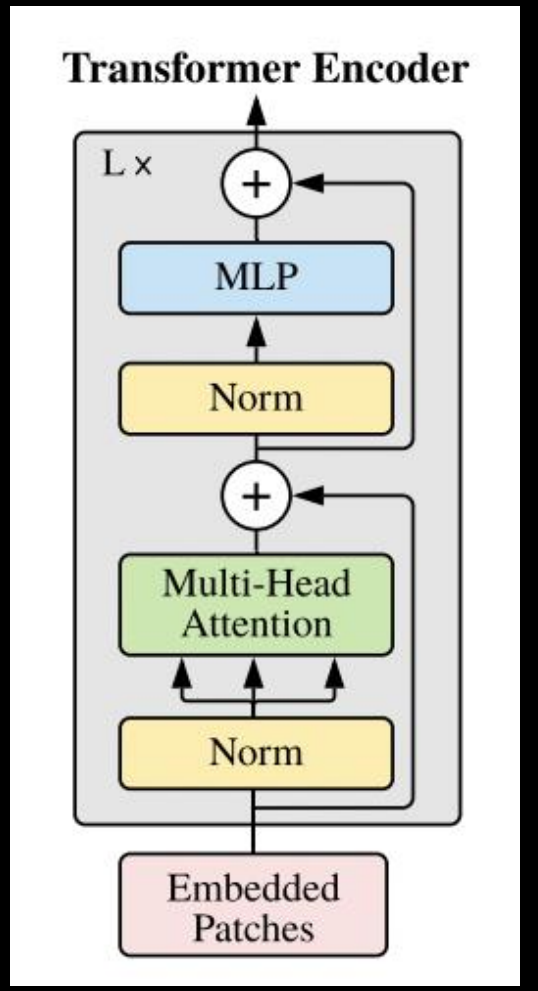


- Embedding
  - Turning words into numbers



- Positional Encoding



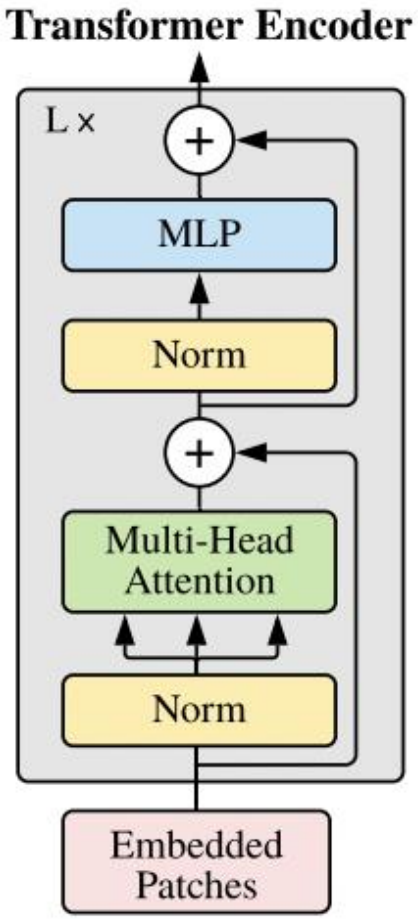


Transformer Encoder Block

- Layer Normalization
  - Applied before each sublayer ("pre-norm" style, used in ViT)
  - Helps stabilize training, especially for deeper models
- Residual Connections
  - Allow gradients to flow more easily and help with convergence
  - Output of each block = input + sublayer output
- Feedforward MLP
  - A 2-layer position-wise MLP is applied to each token independently:

$$\mathrm{MLP}(x) = \mathrm{GELU}(xW_1 + b_1)W_2 + b_2$$

  - Hidden dimension is usually 4× the embedding size (e.g., 768 → 3072 → 768)
  - GELU is the most common activation (better gradient flow than ReLU)



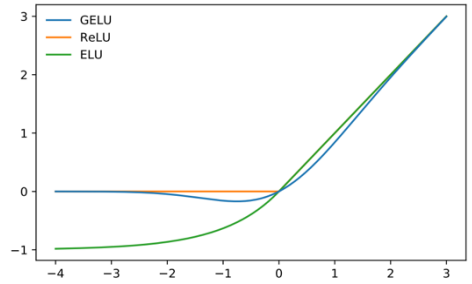


Figure 1: The GELU ($\mu = 0, \sigma = 1$), ReLU, and ELU ($\alpha = 1$).

Transformer Encoder Block

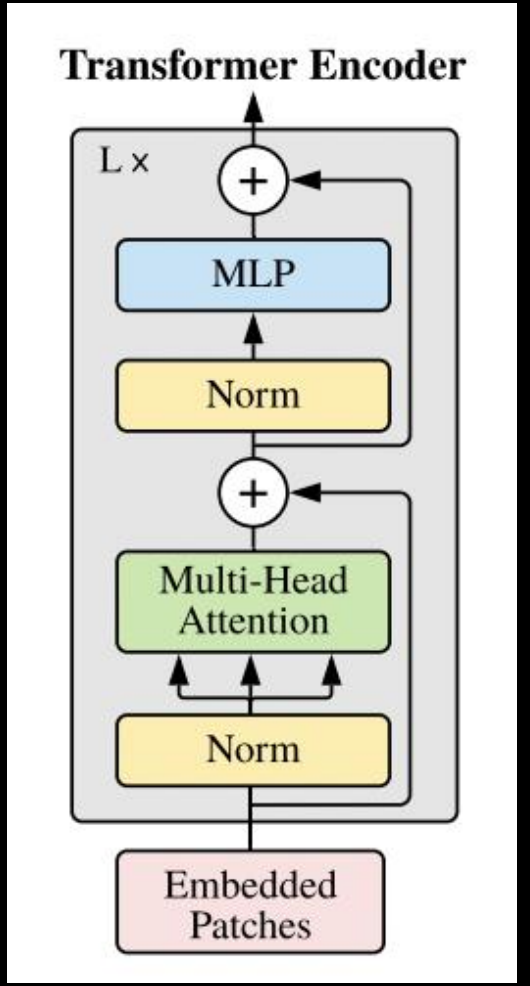
- Multi-Head Attention
  - The self-attention mechanism allows each patch to attend to all other patches, including the [CLS] token. It captures global context, unlike convolutions
  - Compute query (Q), key (K), and value (V) matrices via learned projections:

$$Q = XW^Q, \quad K = XW^K, \quad V = XW^V$$

  - Compute scaled dot-product attention:

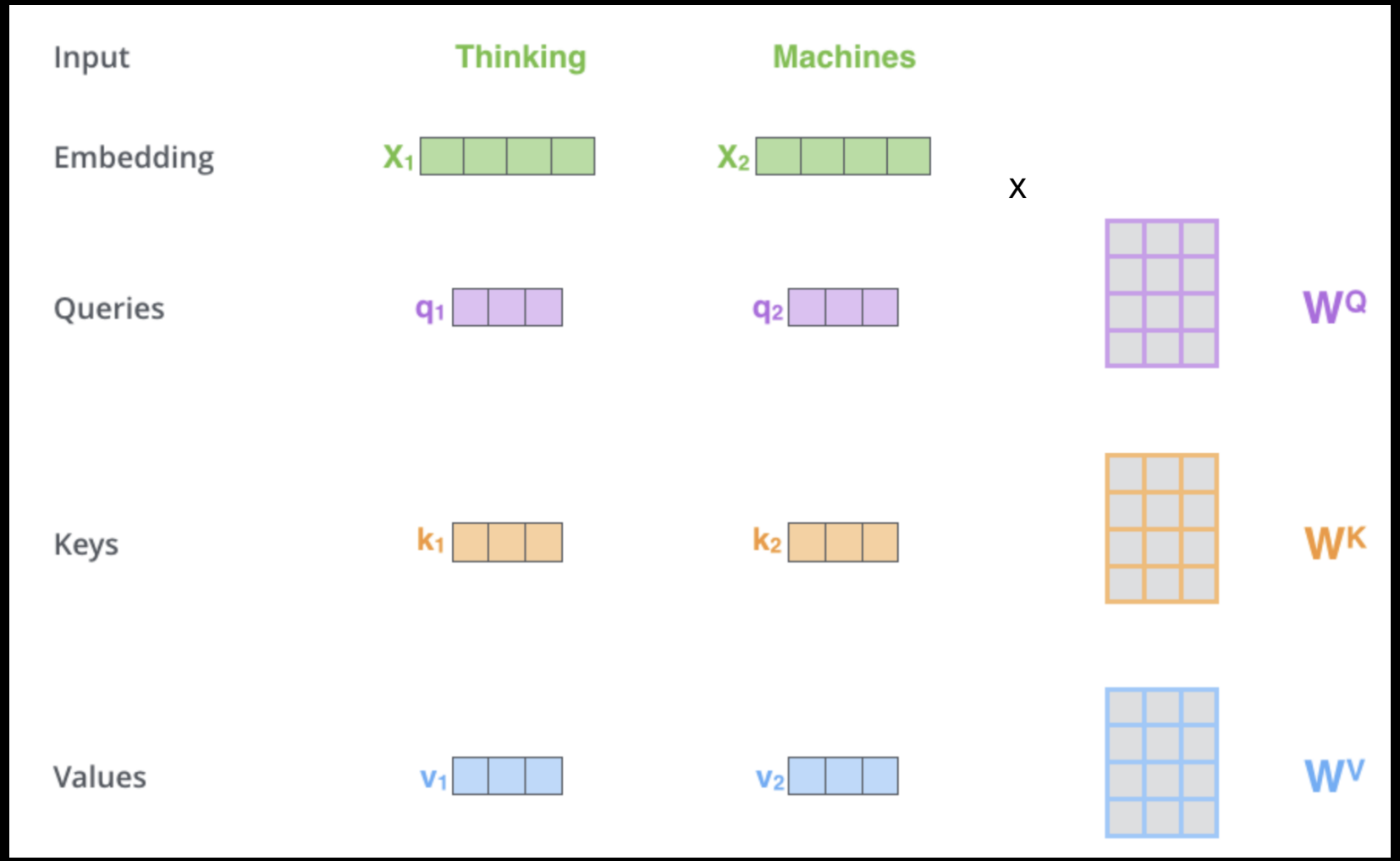
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

  - Each head learns to focus on different types of interactions (e.g., color, shape, location)



Multi-Head Attention in more depth

Create key, query, and value vectors

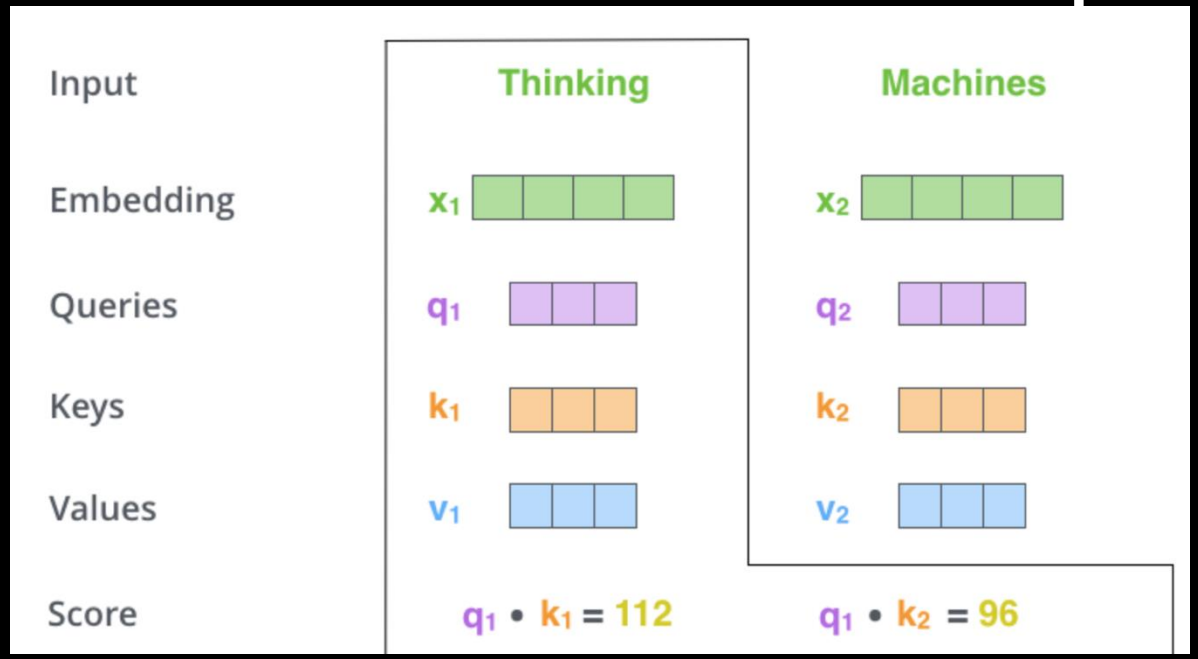


https://jalammar.github.io/illustrated-transformer/

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

Multi-Head Attention in more depth

Dot product of query and key vector



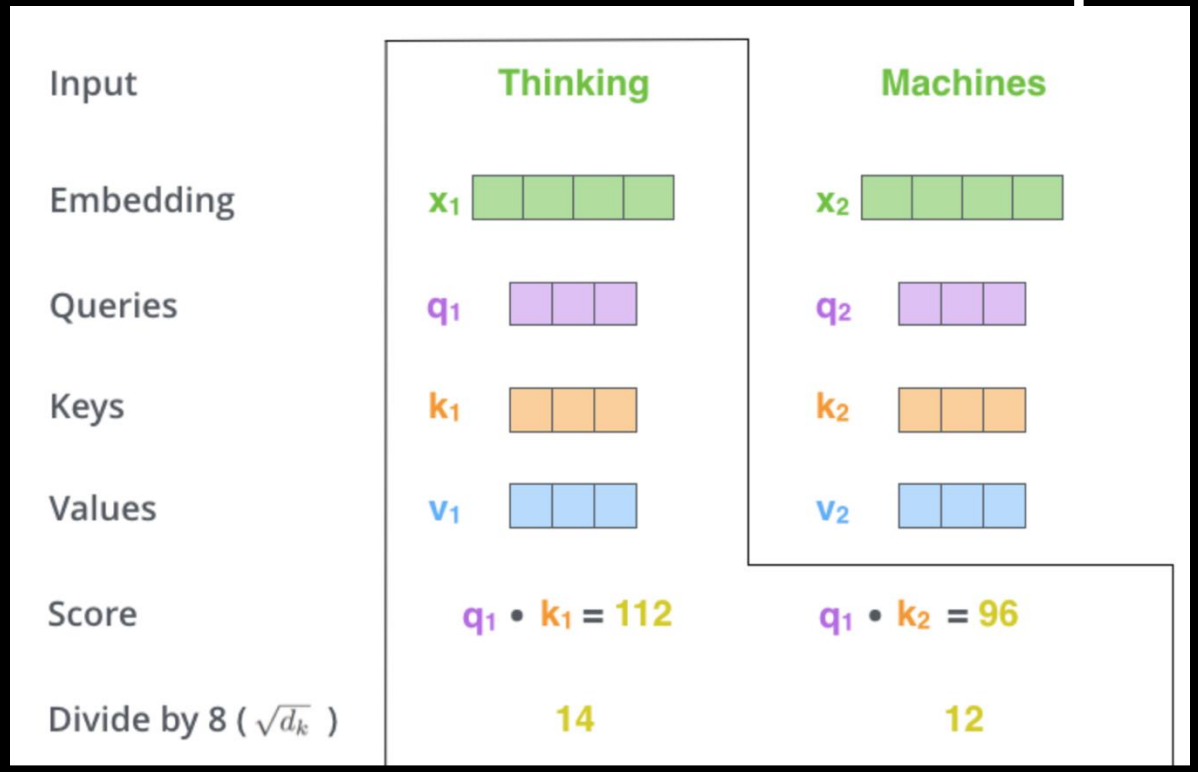
https://jalammar.github.io/illustrated-transformer/

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

Multi-Head Attention in more depth

Dot product of query and key vector

Divide for more stable gradients



the square root of the dimension of the key vectors used in the paper – 64. This leads to having more stable gradients. There could be other possible values here, but this is the default

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

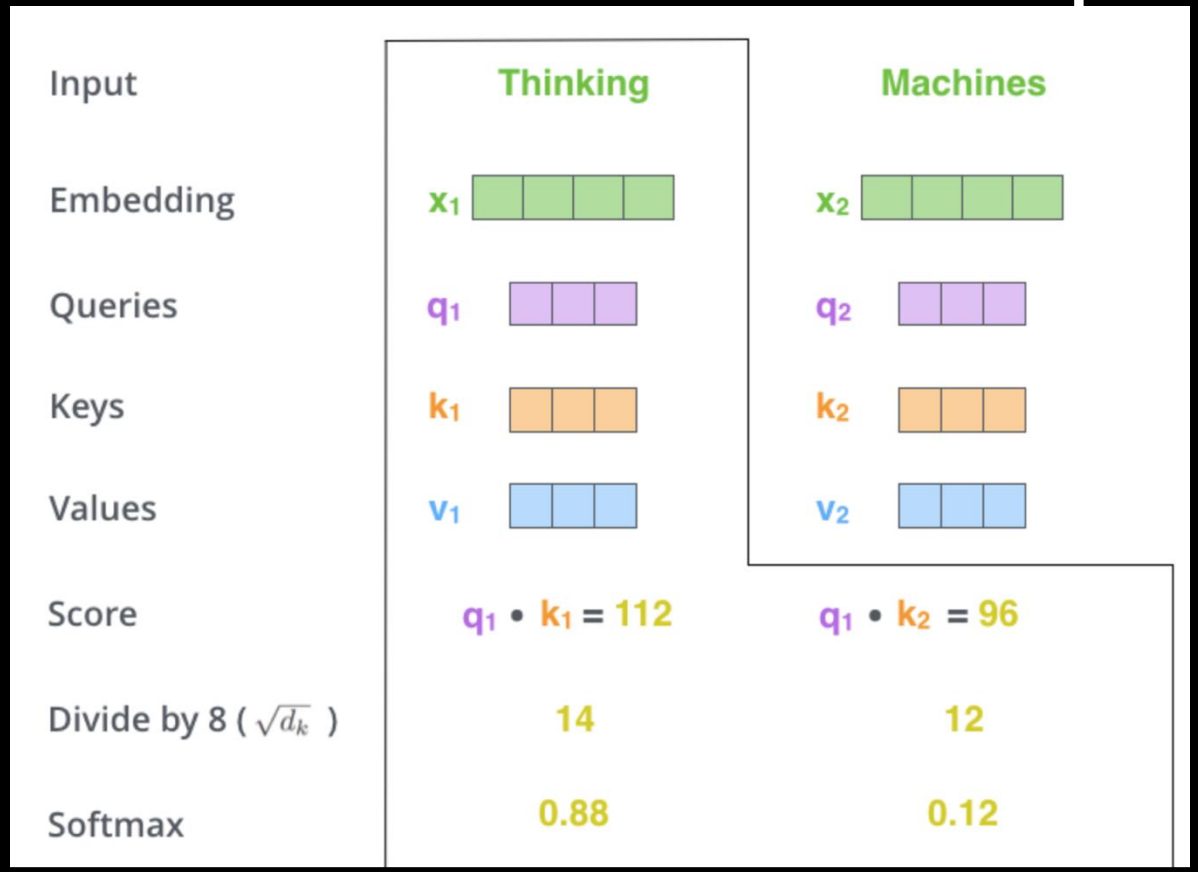
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

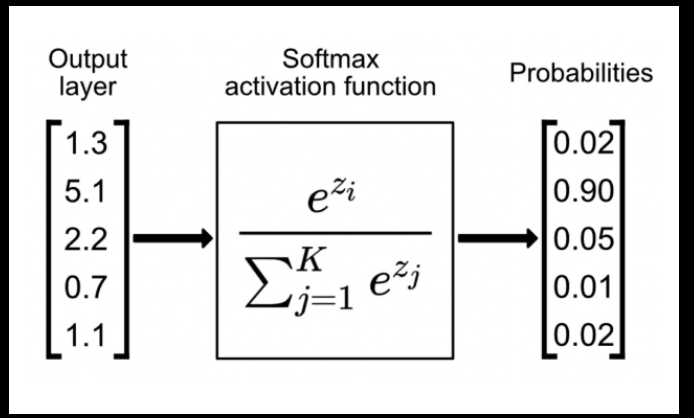
Multi-Head Attention in more depth

Dot product of query and key vector

Divide for more stable gradients

Softmax operation so all positive and sum to 1





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

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

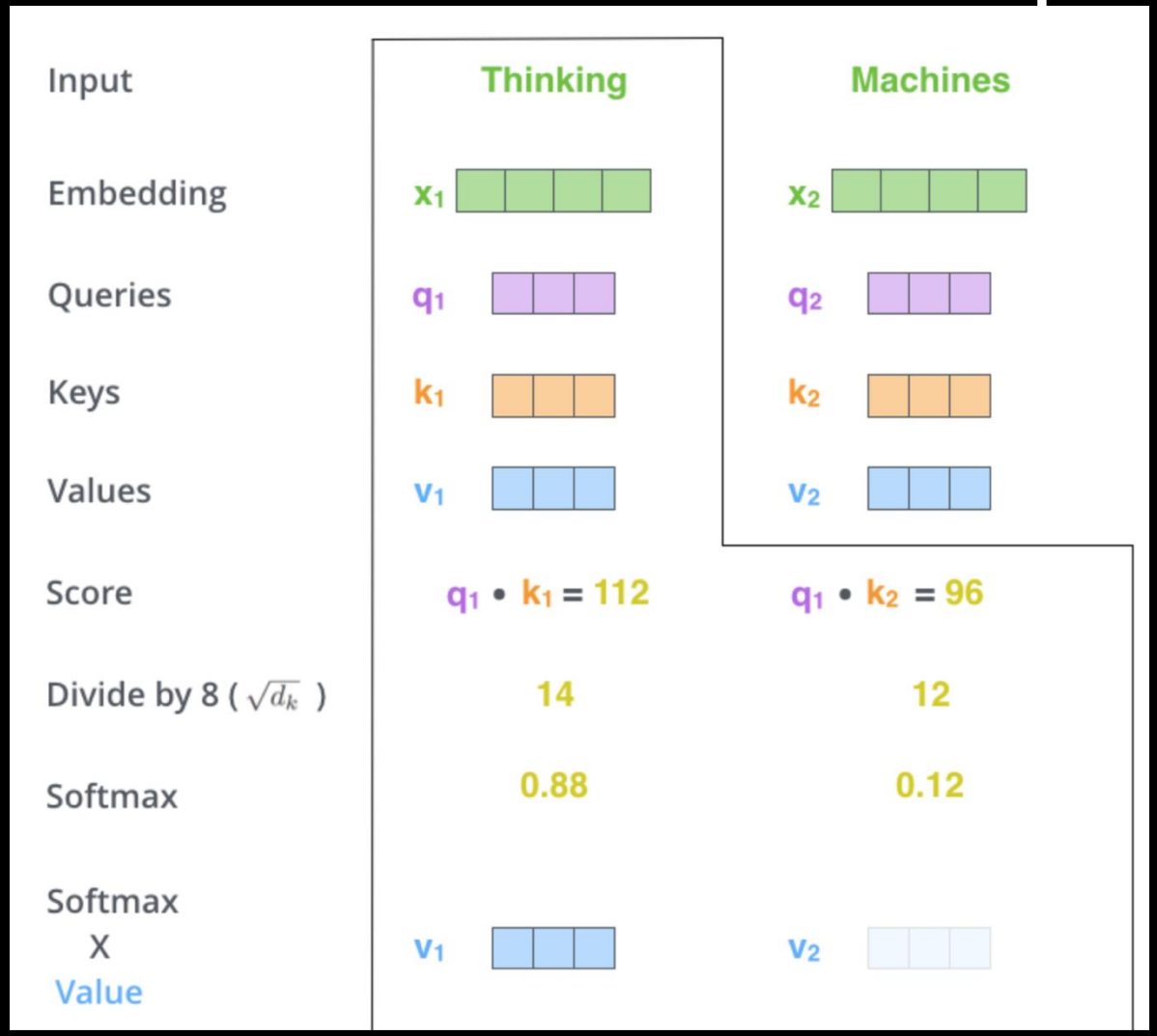
Multi-Head Attention in more depth

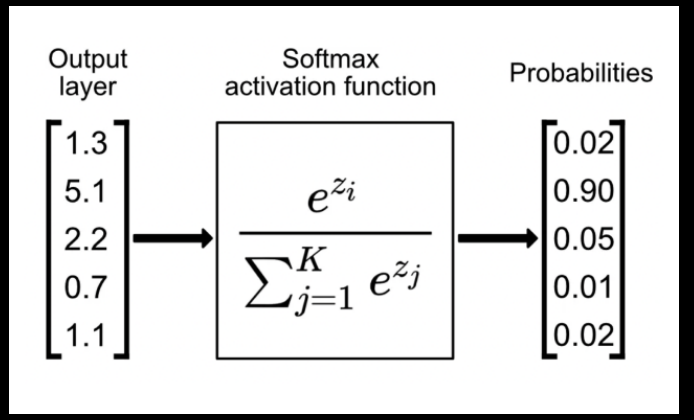
Dot product of query and key vector

Divide for more stable gradients

Softmax operation so all positive and sum to 1

Multiply softmax score to value vector (only keep important words basically)



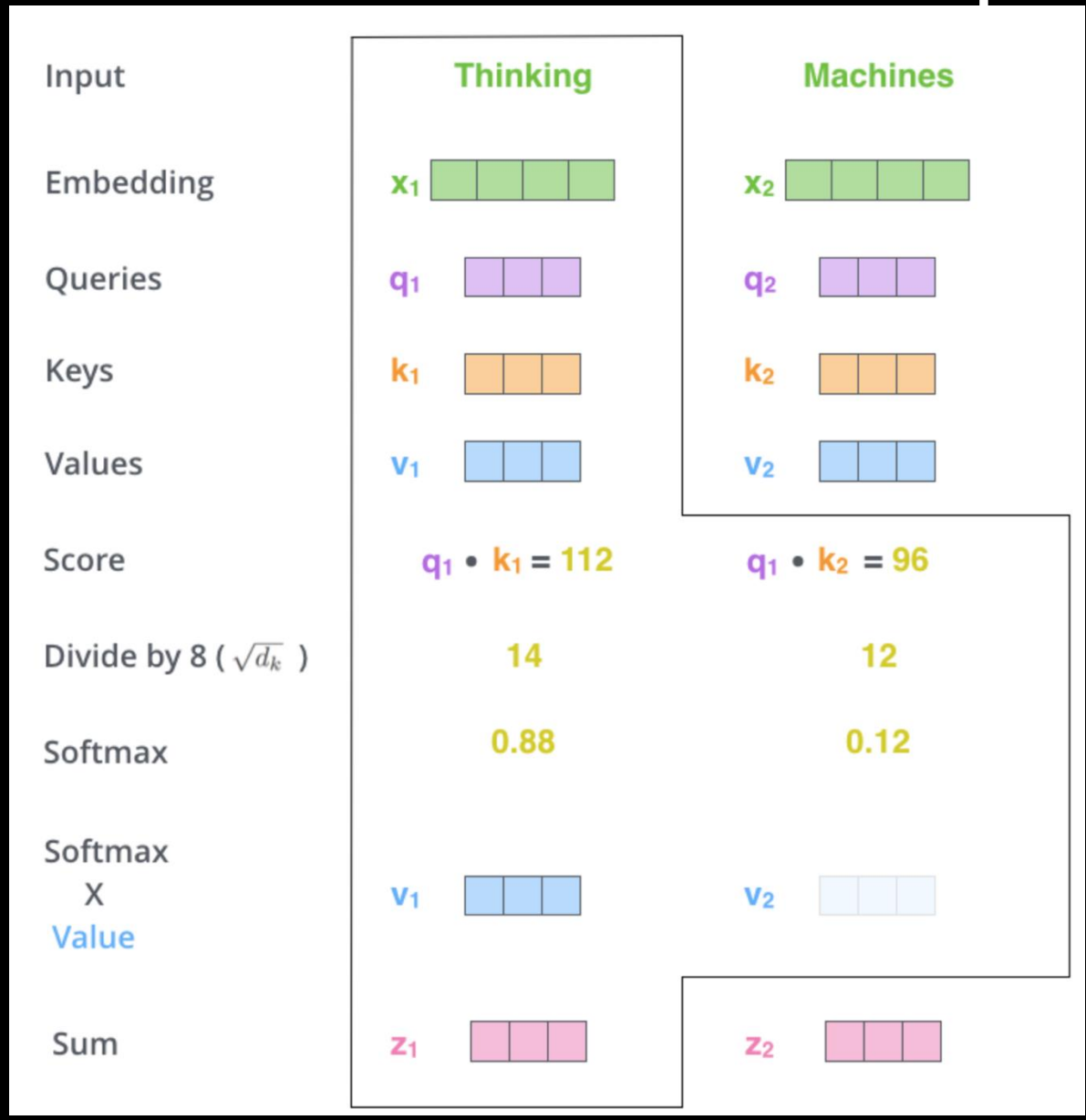


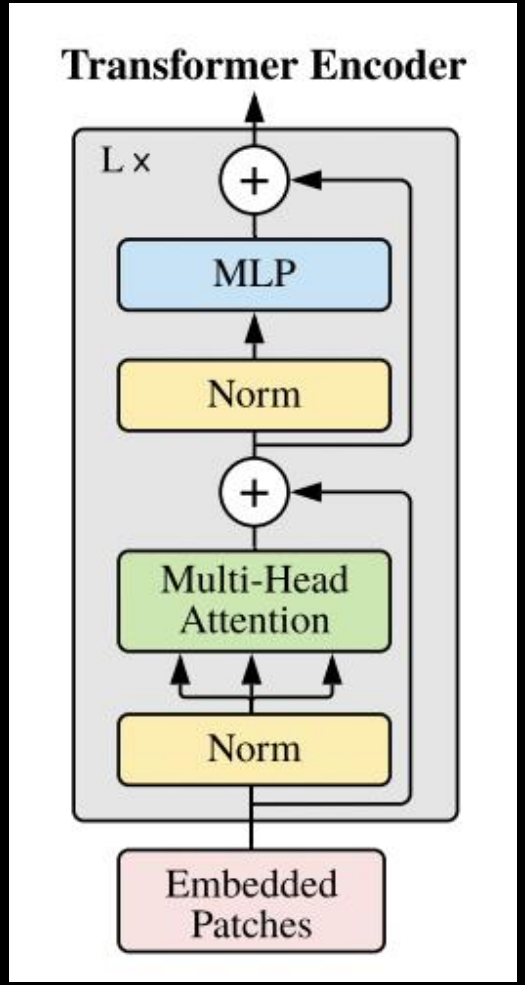
https://jalammar.github.io/illustrated-transformer/

Multi-Head Attention in more depth

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

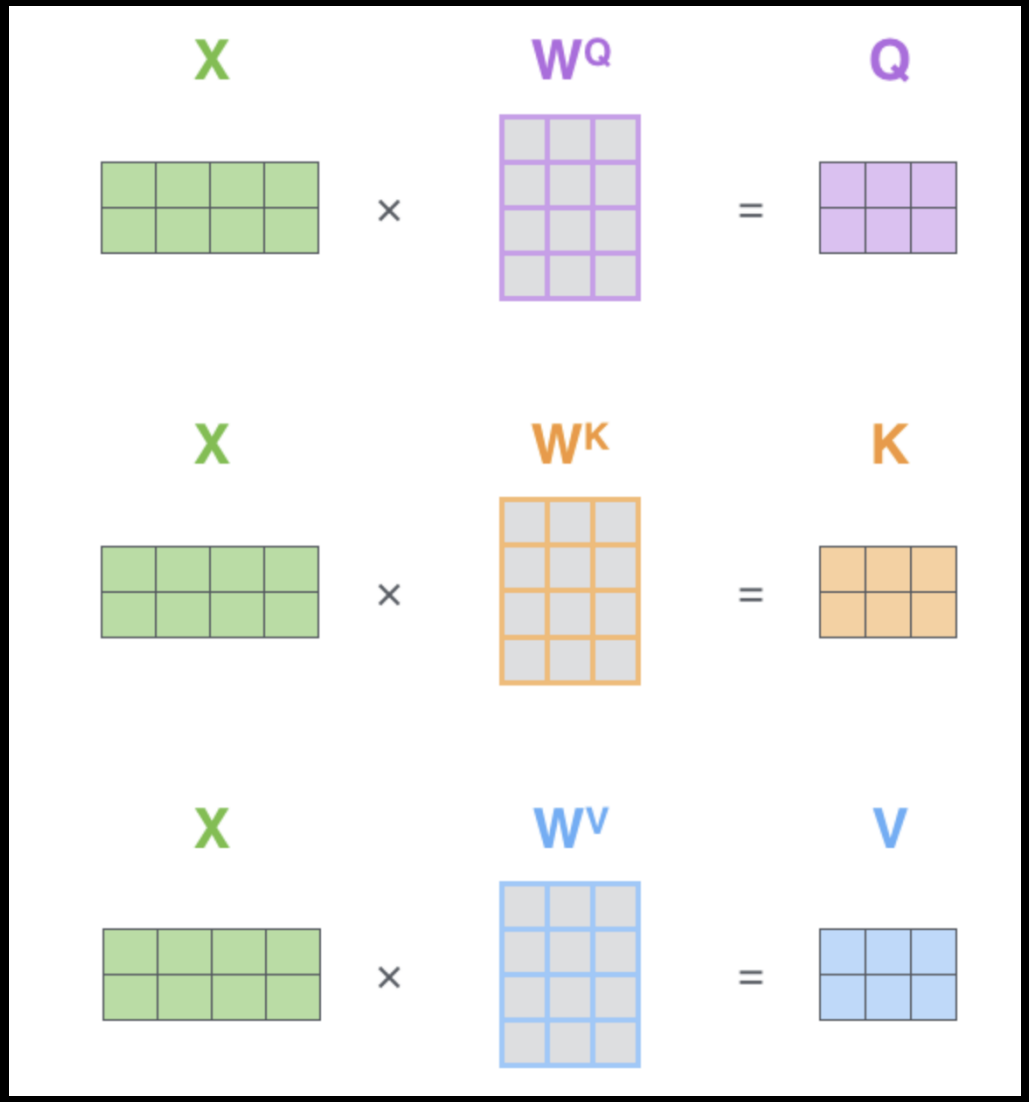
- Dot product of query and key vector
- Divide for more stable gradients
- Softmax operation so all positive and sum to 1
- Multiply softmax score to value vector (only keep important words basically)
- Sum weighted value vectors (input to feedforward)





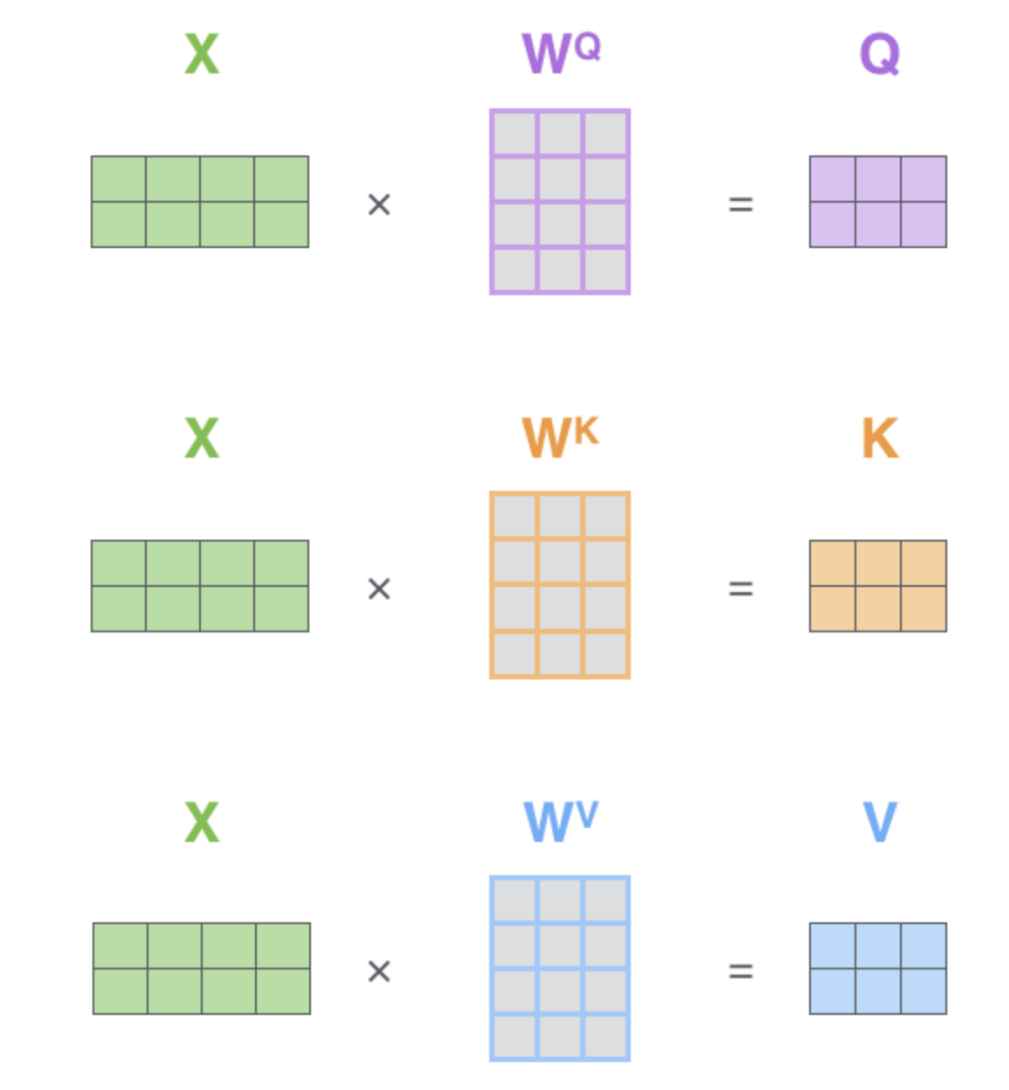
https://jalammar.github.io/illustrated-transformer/

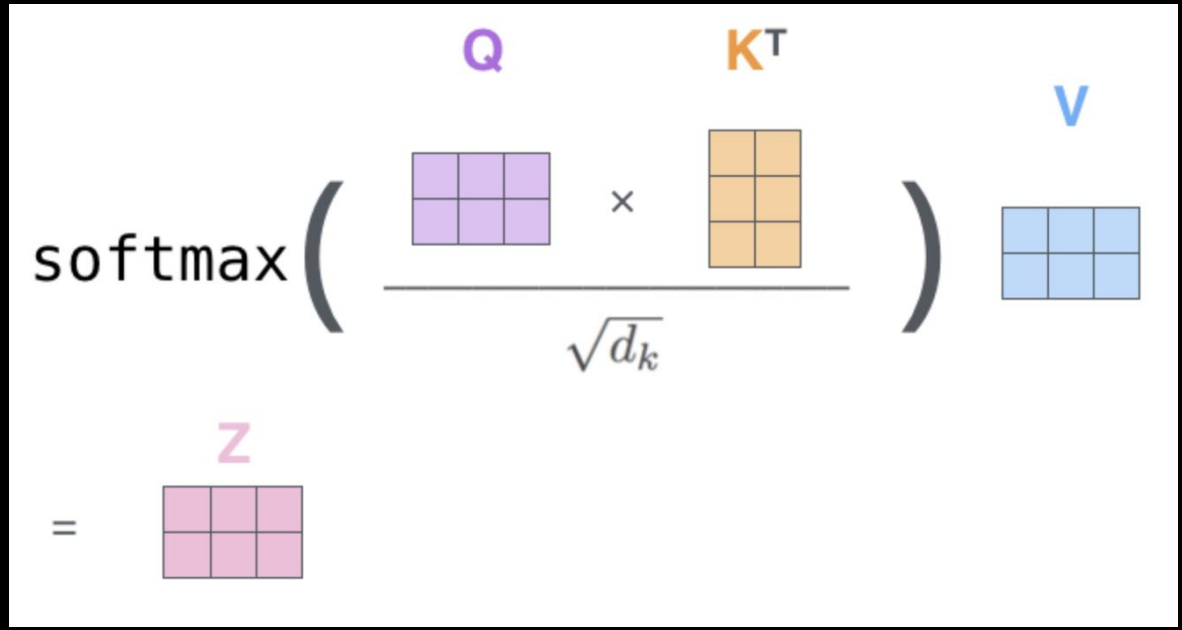
Multi-Head Attention in more depth

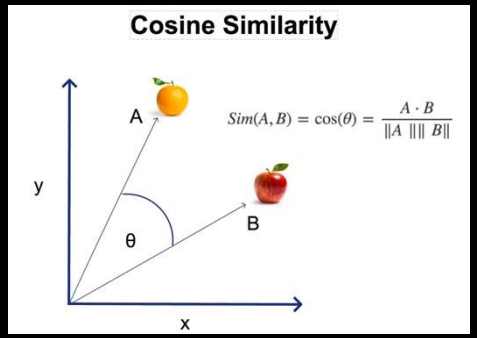


X is word in sentence or patch in image

Multi-Head Attention in more depth

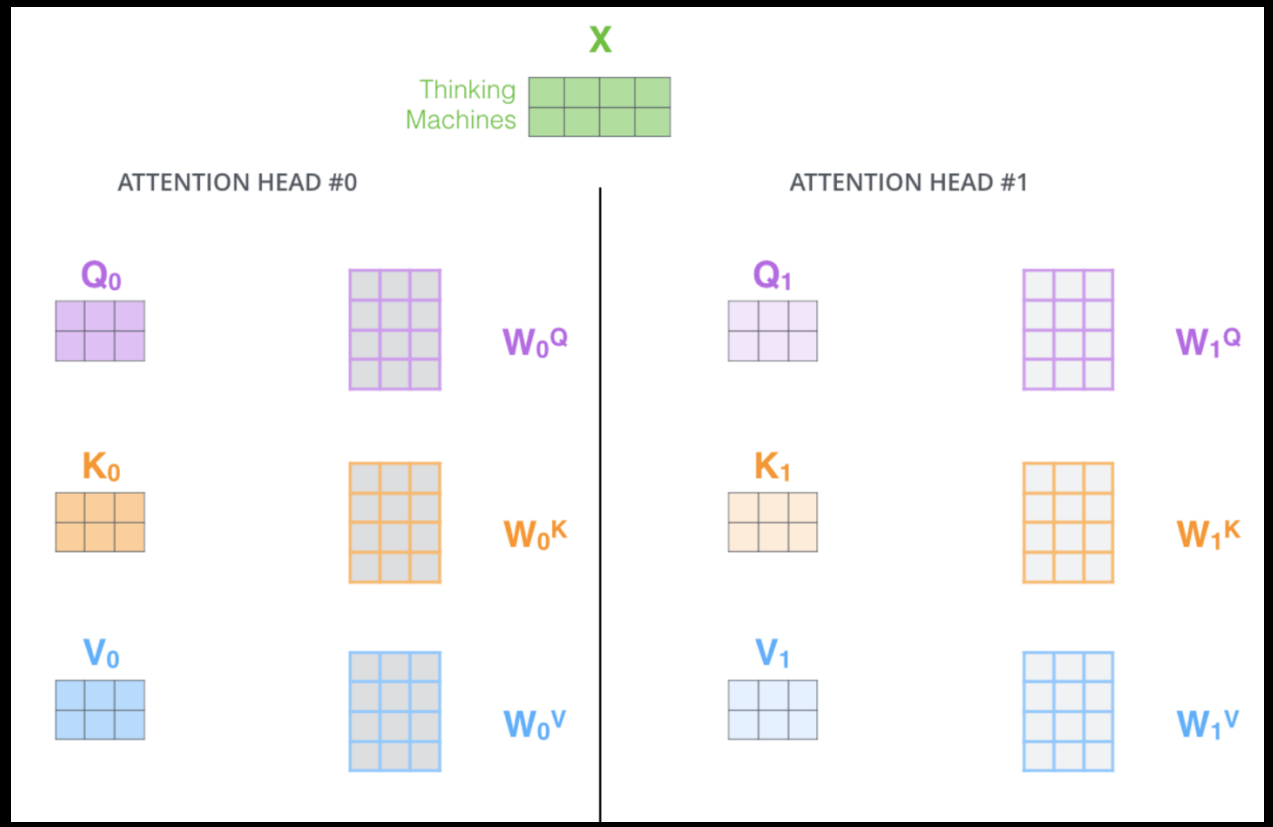




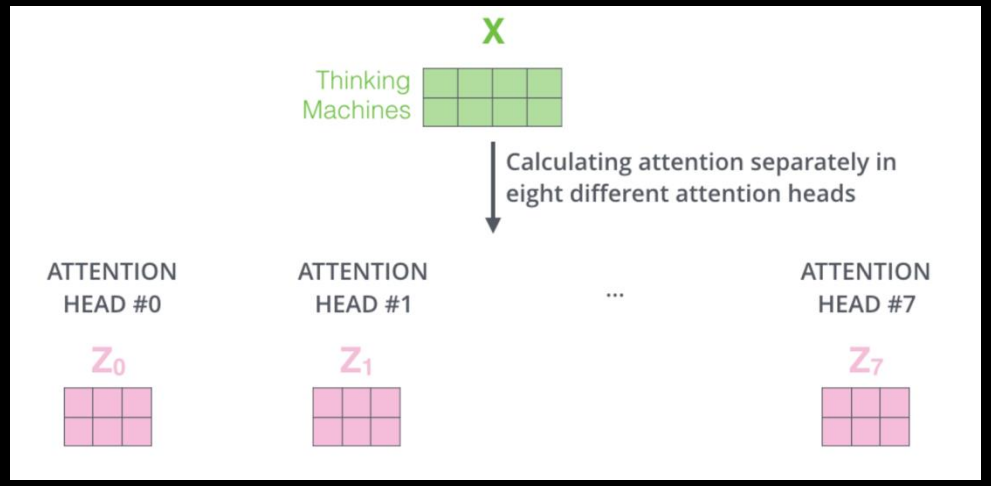


X is word in sentence or patch in image

Multi-Head Attention in more depth



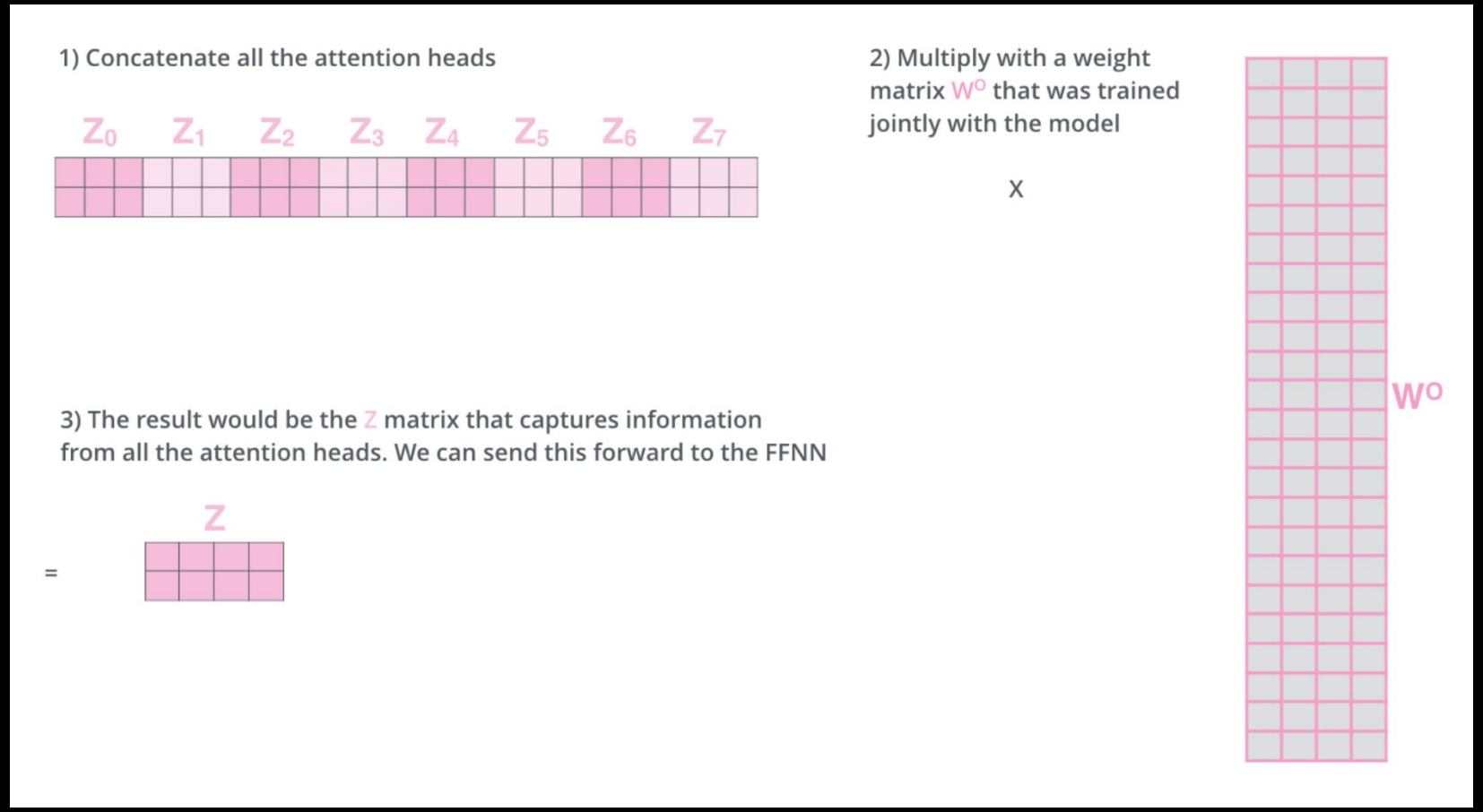
=



Can do multi-headed attention

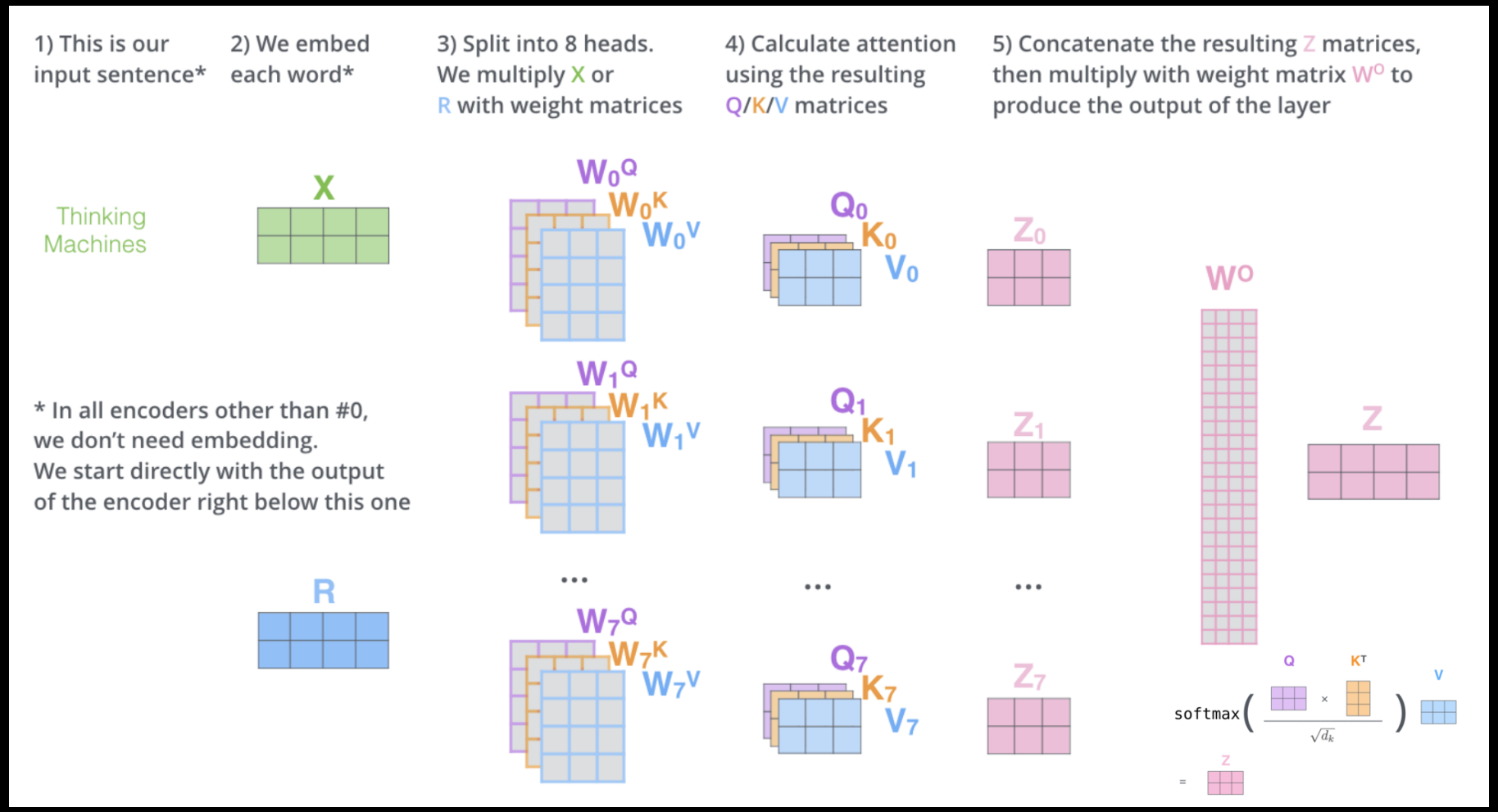
1. It expands the model's ability to focus on different positions
2. It gives the attention layer multiple "representation subspaces". As we'll see next, with multi-headed attention we have not only one, but multiple sets of Query/Key/Value weight matrices (the Transformer uses eight attention heads, so we end up with eight sets for each encoder/decoder). Each of these sets is randomly initialized.

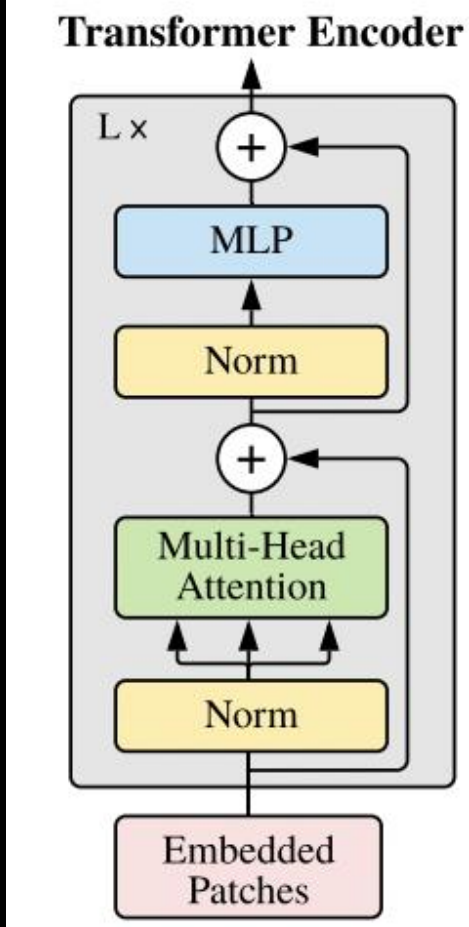
Multi-Head Attention in more depth



So just concatenate matrices and multiply by another weight matrix

Multi-Head Attention in more depth





The Decoder Side

- The output of the top encoder is then transformed into a set of attention vectors K and V. These are to be used by each decoder in its “encoder-decoder attention” layer which helps the decoder focus on appropriate places in the input sequence

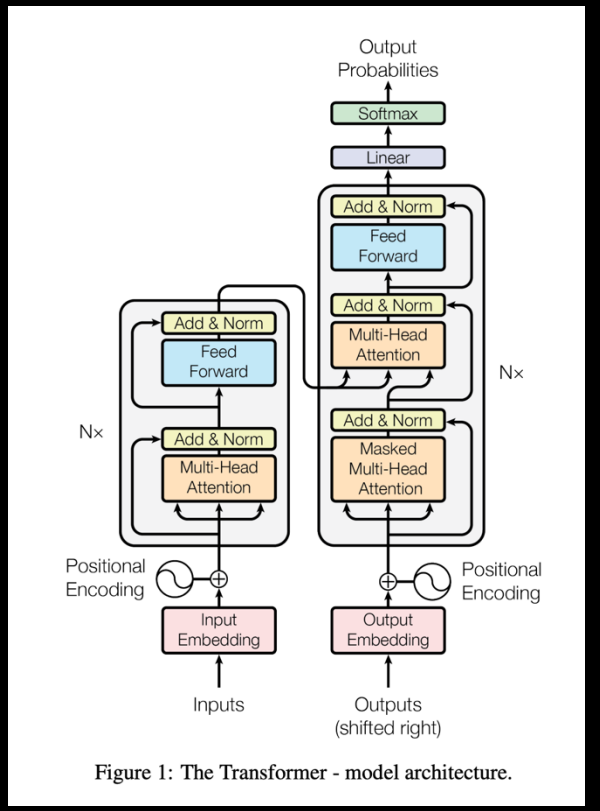
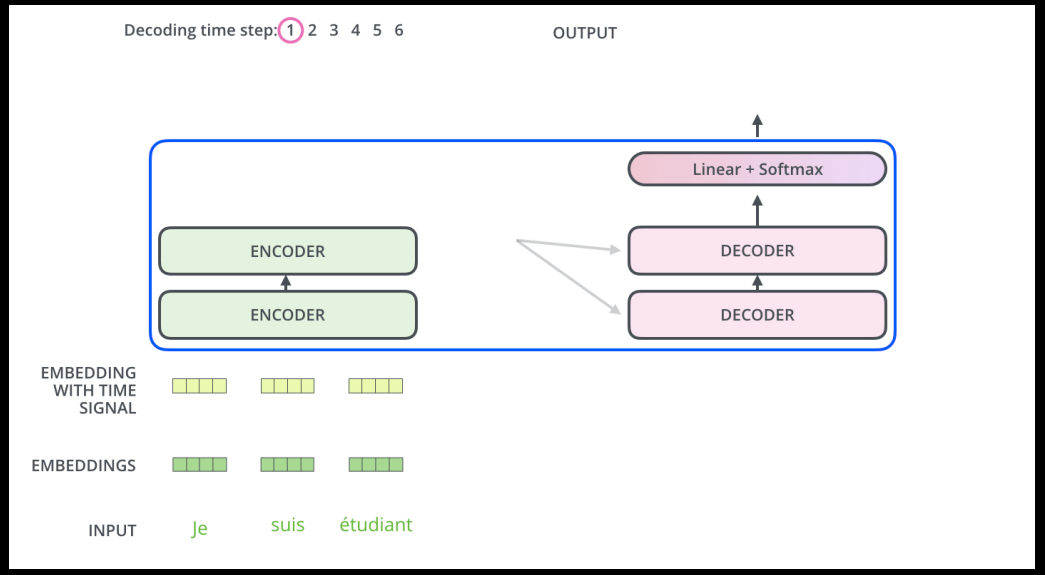


Figure 1: The Transformer - model architecture.



The Decoder Side

- The output of the top encoder is then transformed into a set of attention vectors K and V. These are to be used by each decoder in its “encoder-decoder attention” layer which helps the decoder focus on appropriate places in the input sequence
- The following steps repeat the process until a special symbol is reached indicating the transformer decoder has completed its output. The output of each step is fed to the bottom decoder in the next time step, and the decoders bubble up their decoding results just like the encoders did.

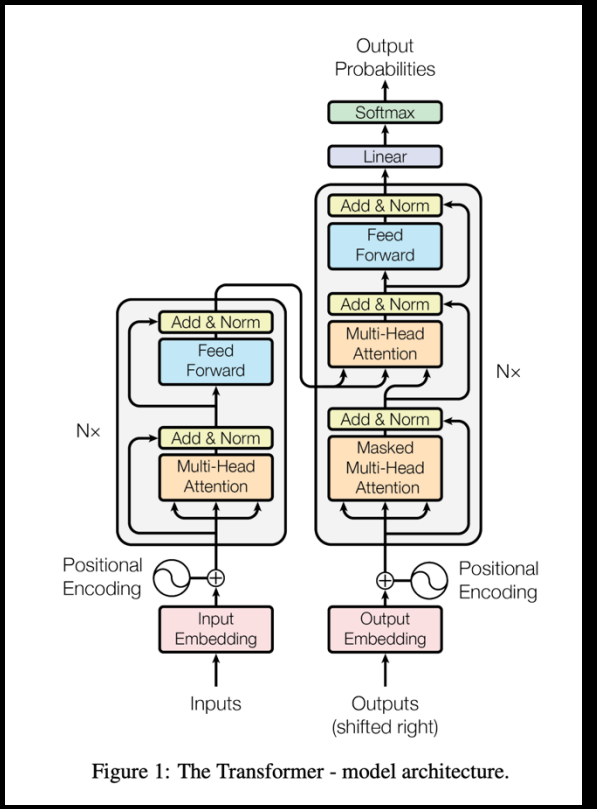
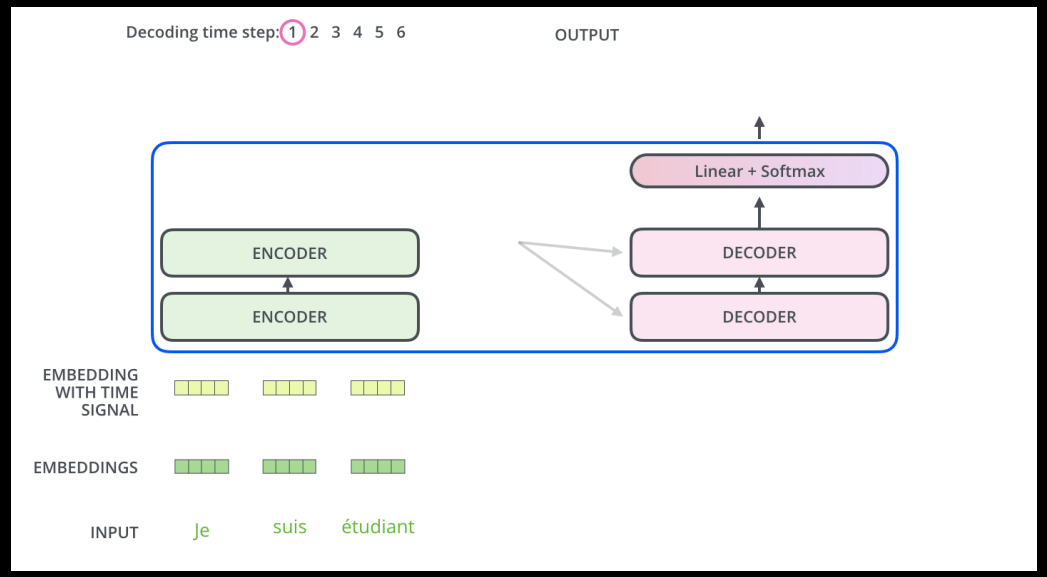
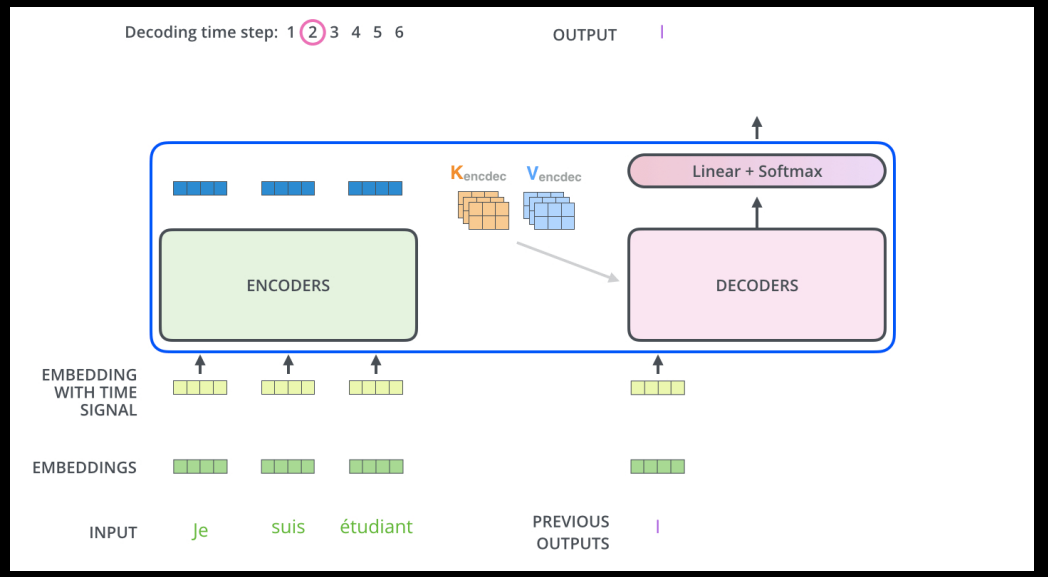


Figure 1: The Transformer - model architecture.

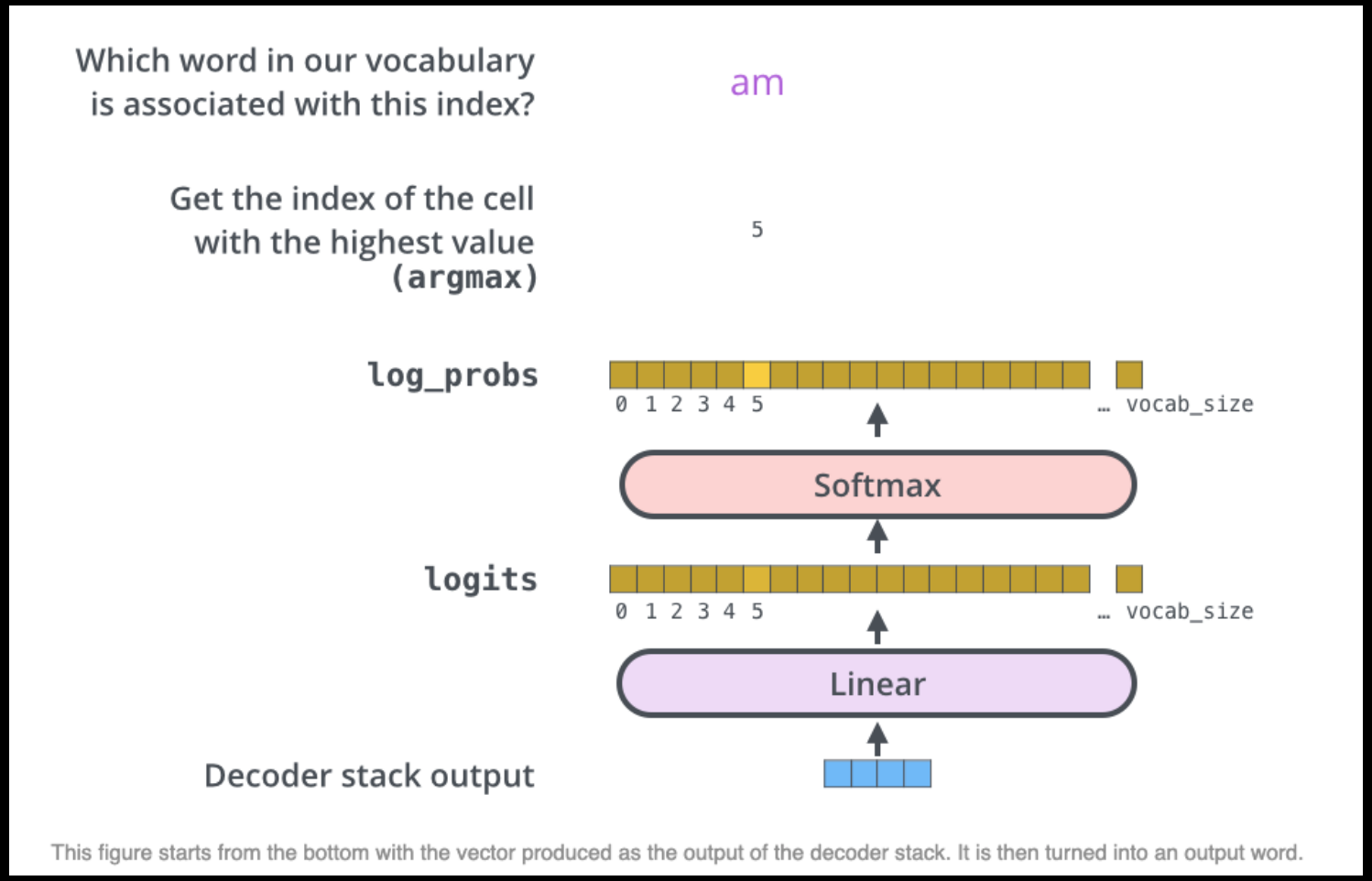




Main difference: the self-attention layer is only allowed to attend to earlier positions in the output sequence

The Decoder Side

- The decoder stack outputs a vector of floats that is put into a Linear and Softmax layer



This figure starts from the bottom with the vector produced as the output of the decoder stack. It is then turned into an output word.

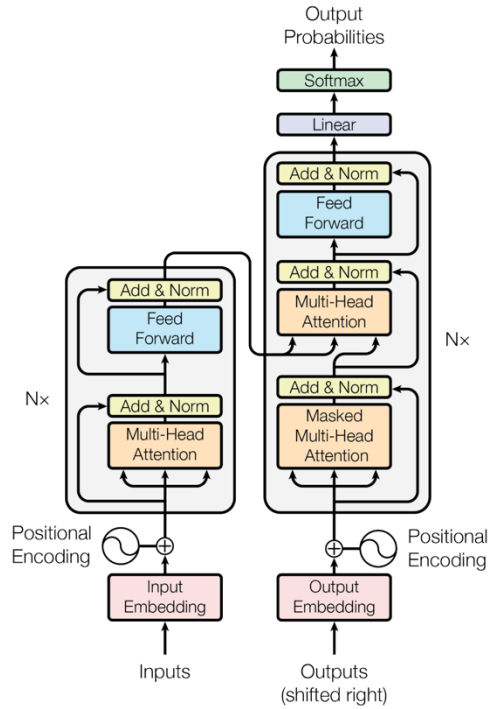
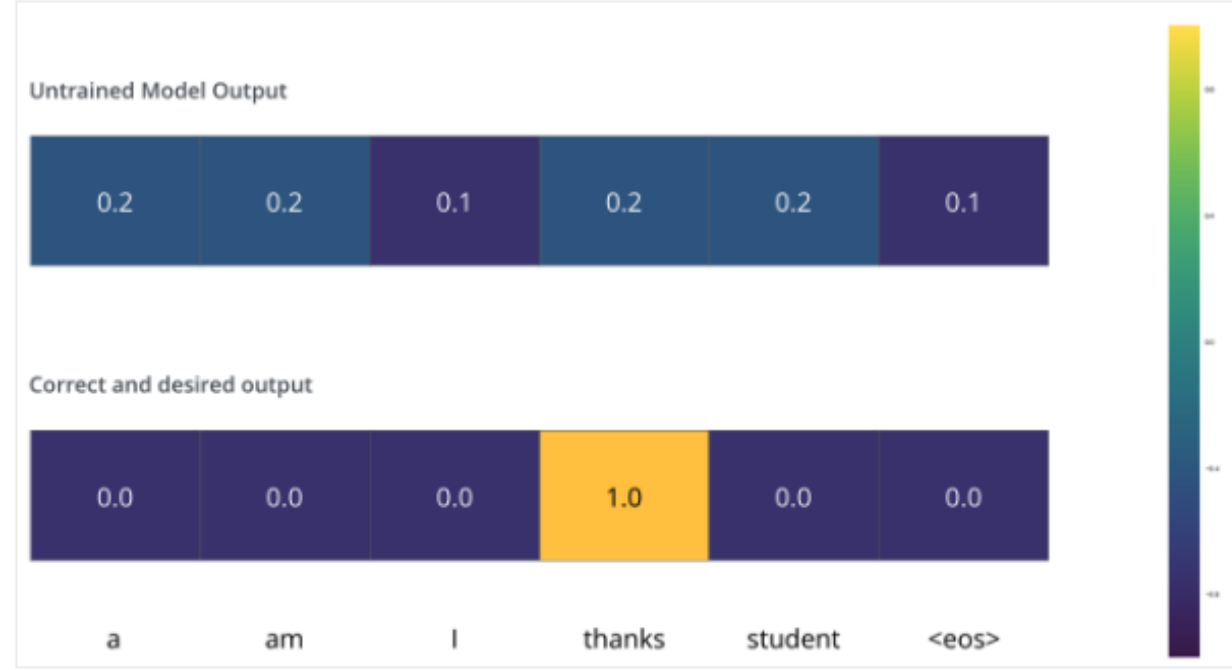


Figure 1: The Transformer - model architecture.

The Loss Function

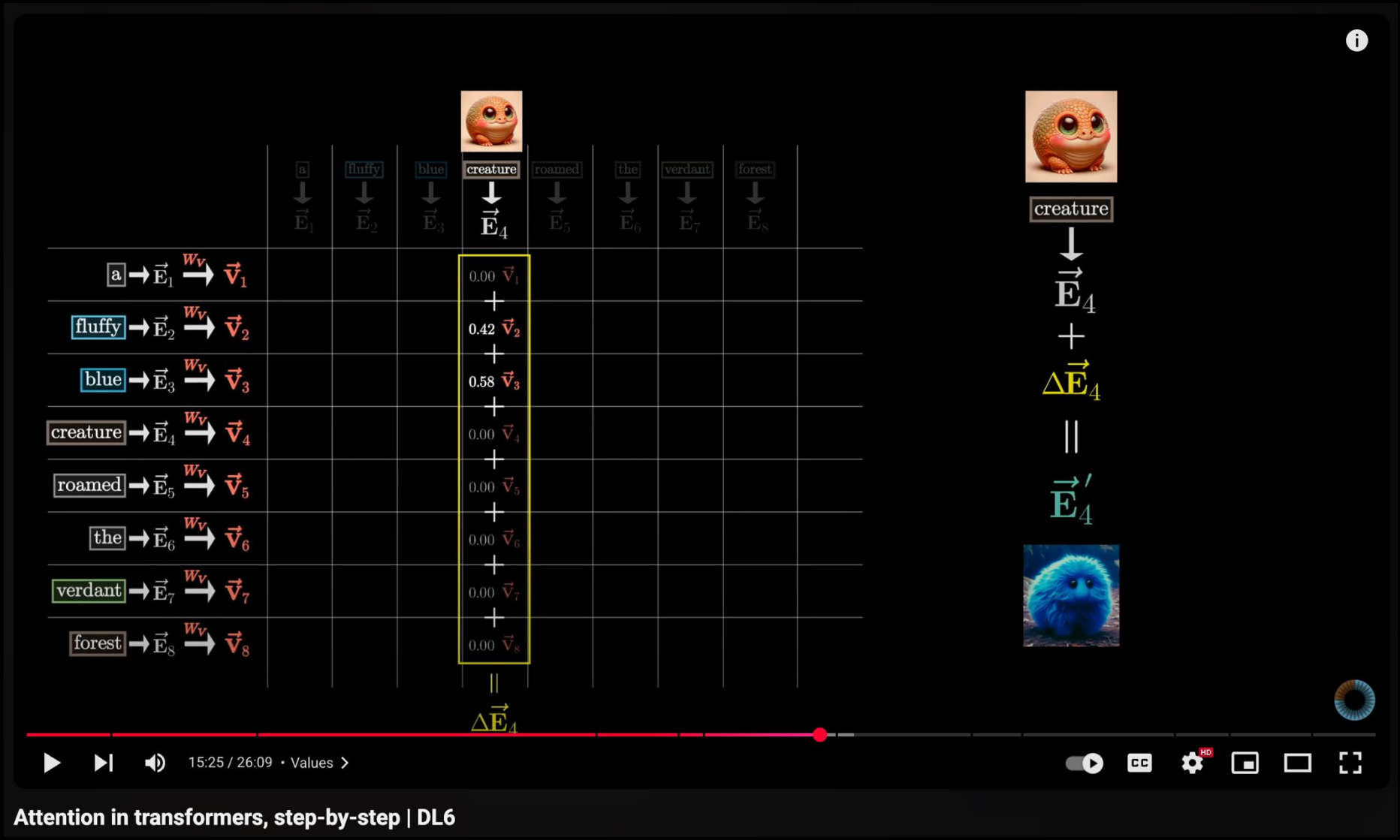
- Say we want to translate ‘merci’ to ‘thanks’
- We can use cross-entropy or Kullback-Leibler divergence to compare two probability to act as our loss function



Since the model's parameters (weights) are all initialized randomly, the (untrained) model produces a probability distribution with arbitrary values for each cell/word. We can compare it with the actual output, then tweak all the model's weights using backpropagation to make the output closer to the desired output.

Check out 3Blue1Brown for a more intuitive explanation!

https://www.youtube.com/@3blue1brown



Attention in transformers, step-by-step | DL6

https://www.youtube.com/watch?v=eMlx5fFNoYc&t=858s

Transformers for Vision

- Vision tasks also benefit from global context (e.g., understanding an object requires global shape, not just local texture)
- CNNs struggle with global reasoning and scaling to larger datasets without architectural tweaks
- Vision Transformers aim to:
  - Replace convolutions with attention
  - Use minimal inductive bias and let the model learn from data
  - Scale better with data and compute

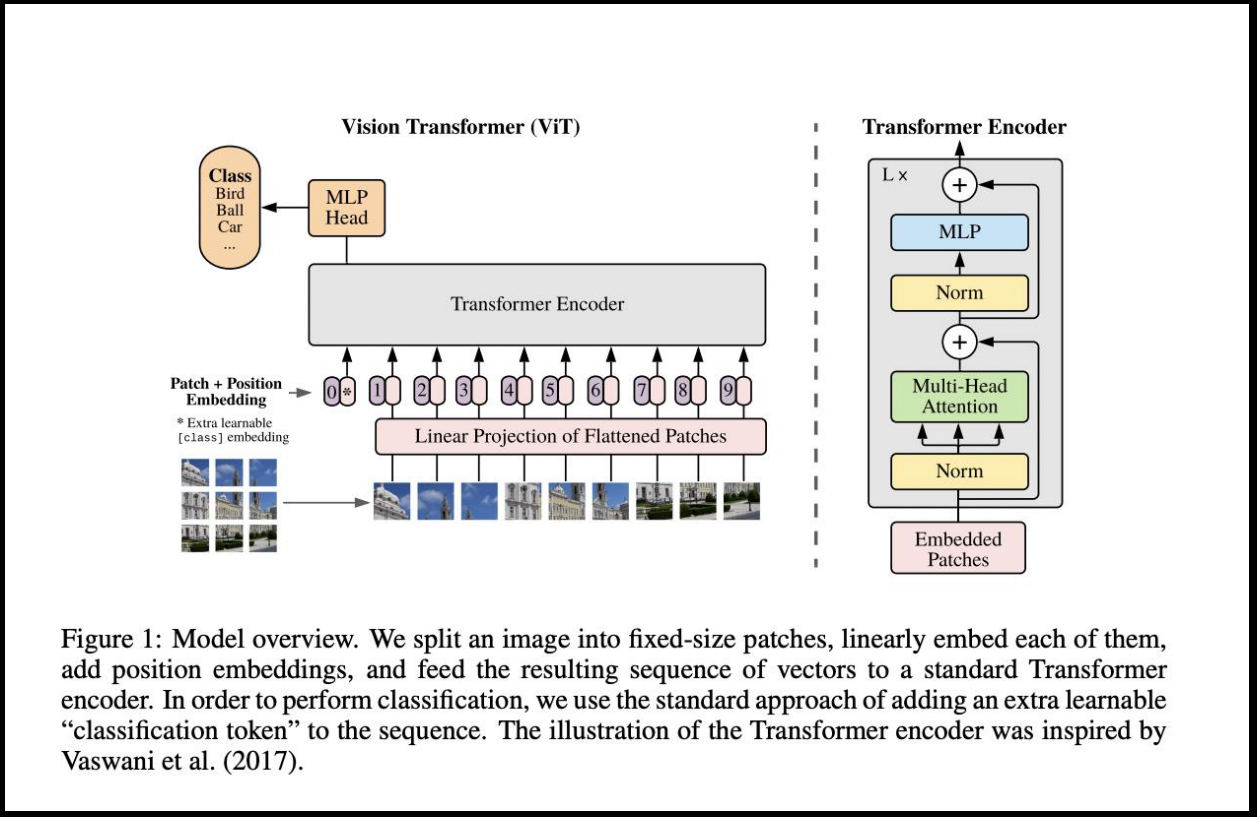


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification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

https://arxiv.org/pdf/2010.11929

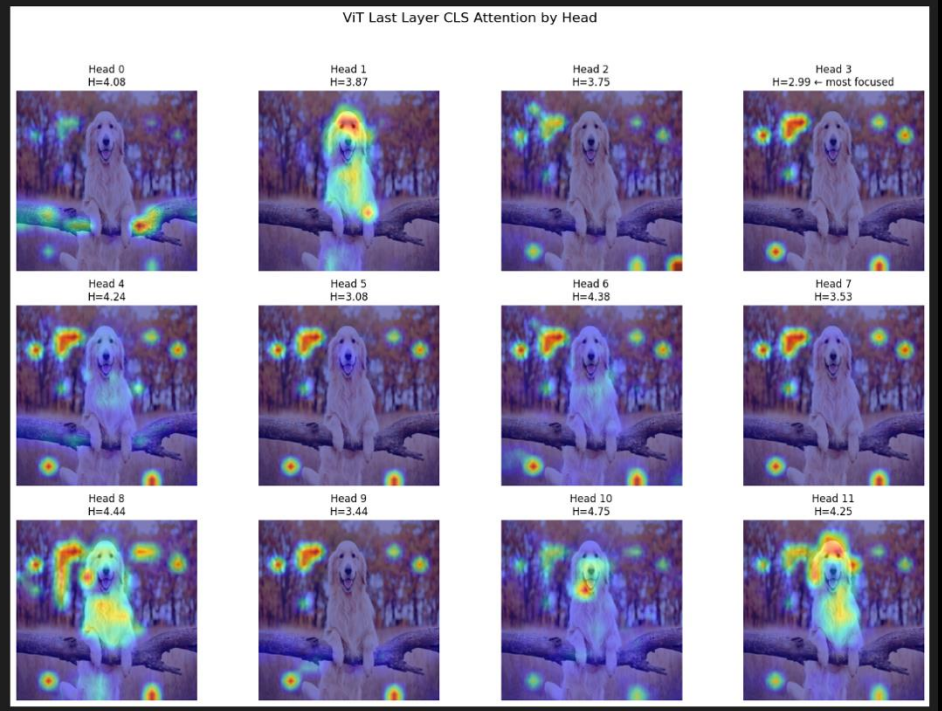
Examples of Attention in Images

”Attention maps refer to the visualizations of the attention weights that are calculated between each token (or patch) in the image and all other tokens.”

1. Take class token’s attention matrix row
2. Reshape to dimension of image
3. Plot

```
Top 5 predictions:
  golden retriever     0.8920
  Labrador retriever   0.0810
  kuvasz               0.0036
  Great Pyrenees       0.0016
  Chesapeake Bay retriever 0.0012

Head with lowest entropy (most focused): Head 3 (H=2.994)
```



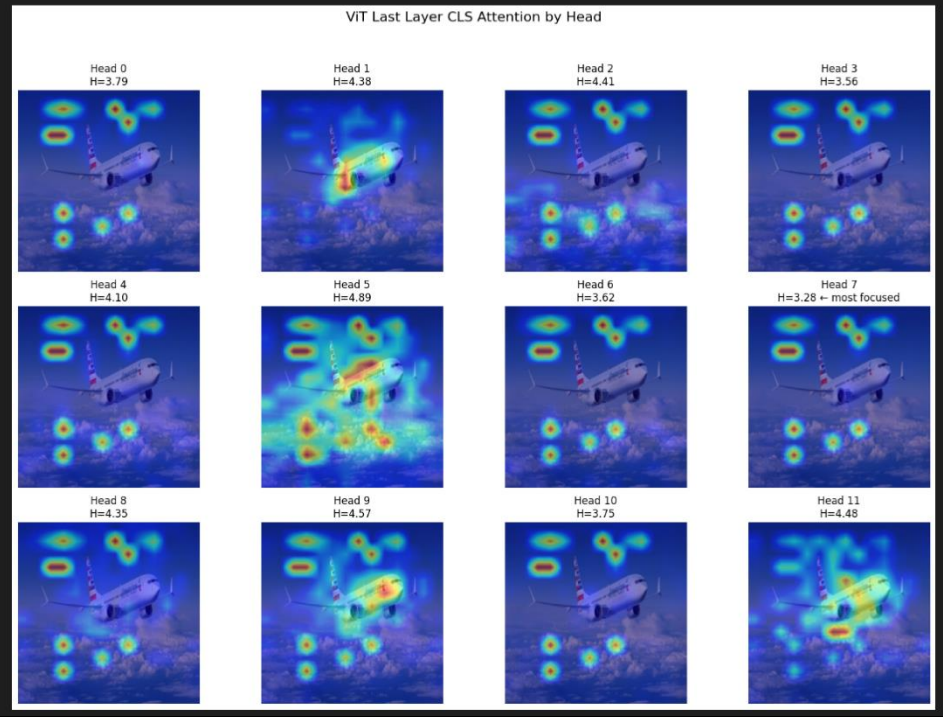
Examples of Attention in Images

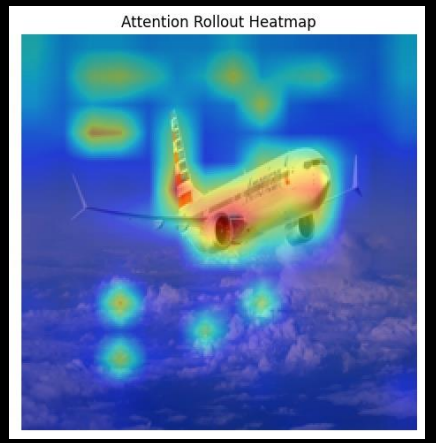
”Attention maps refer to the visualizations of the attention weights that are calculated between each token (or patch) in the image and all other tokens.”

1. Take class token’s attention matrix row
2. Reshape to dimension of image
3. Plot


```
Top 5 predictions:
  airliner             0.9747
  wing                 0.0197
  warplane, military plane 0.0018
  space shuttle        0.0009
  airship, dirigible   0.0003

Head with lowest entropy (most focused): Head 7 (H=3.275)
```





Downsides of Transformers

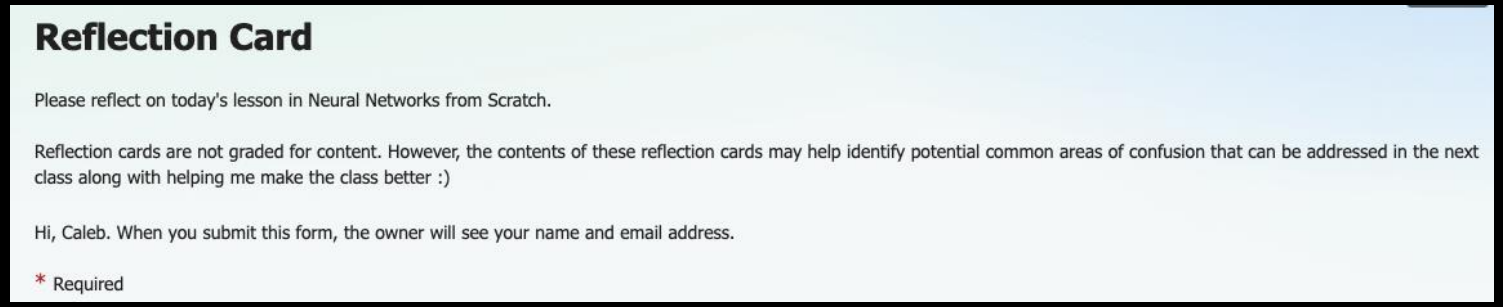
- Need a TON of data to train robustly
  - Quality of data becomes much more important
  - Must learn local patterns without being told unlike CNNs
  - Must learn from multiple examples translational invariance
  - “No free lunch—if you remove the “helpful shortcuts” (inductive biases), a model must see more data to rediscover them from examples alone. That’s why pure Vision Transformers are so data-hungry on small or moderately sized image sets.”

- O(N^2) self attention is expensive for memory and computing

Transformers Visualized

- https://poloclub.github.io/transformer-explainer/

Reflection Cards



Reflection Card

Please reflect on today's lesson in Neural Networks from Scratch.

Reflection cards are not graded for content. However, the contents of these reflection cards may help identify potential common areas of confusion that can be addressed in the next class along with helping me make the class better :)

Hi, Caleb. When you submit this form, the owner will see your name and email address.

* Required

- Essentially a means to help me make this class better!

1. What is something that you learned in today's lecture?
2. What is something that you are still confused about from today's lecture?
3. Do you have any other comments/feedback/thoughts/suggestions/concerns?

https://forms.office.com/r/Kv8LtW4iJH