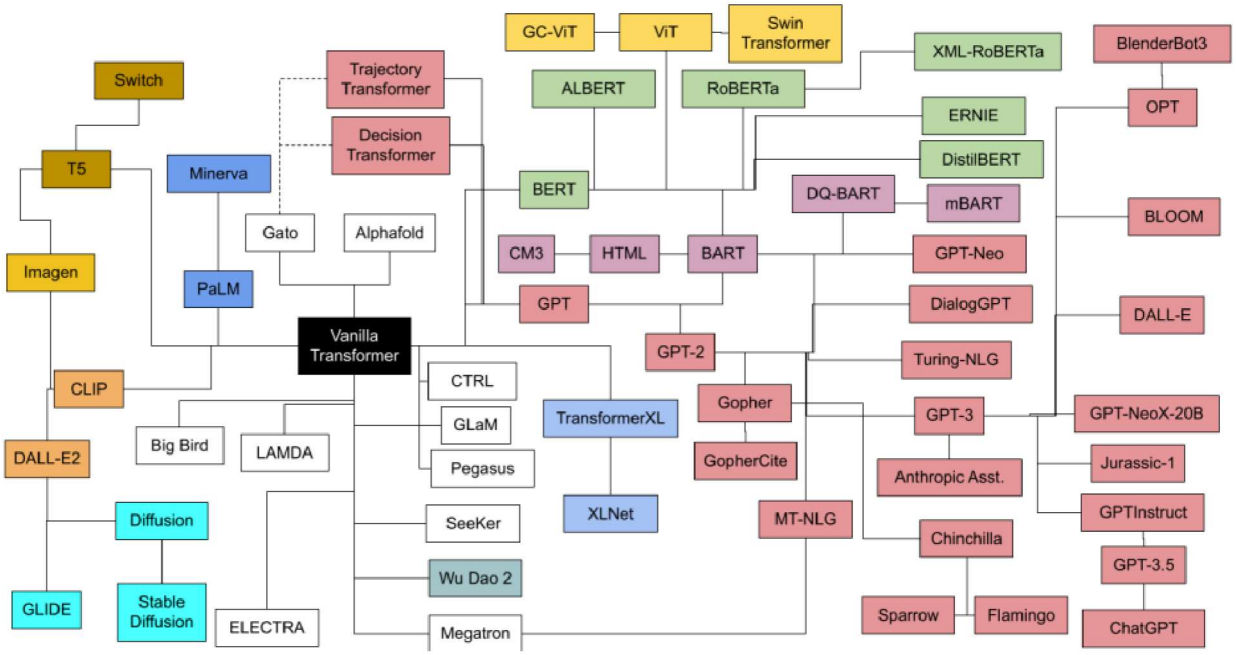
Session 16 - Introduction to Transformers - Part I

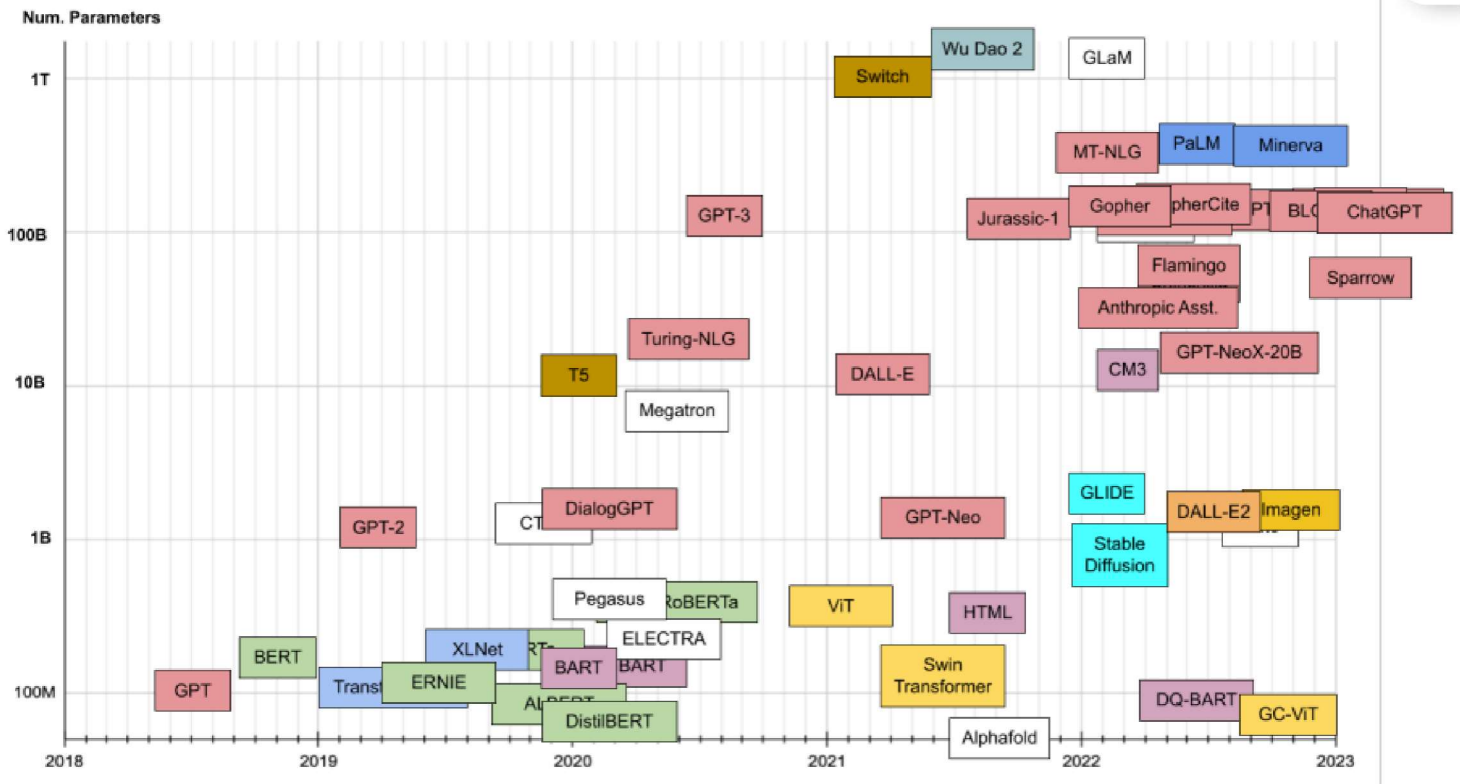
- Due Saturday by 9am
- Points 0
- Available after May 18 at 11am

Transformer models have recently demonstrated exemplary performance on a broad range of language tasks, e.g. text classification, machine translation, and question answering. Among these models, the most popular ones include:

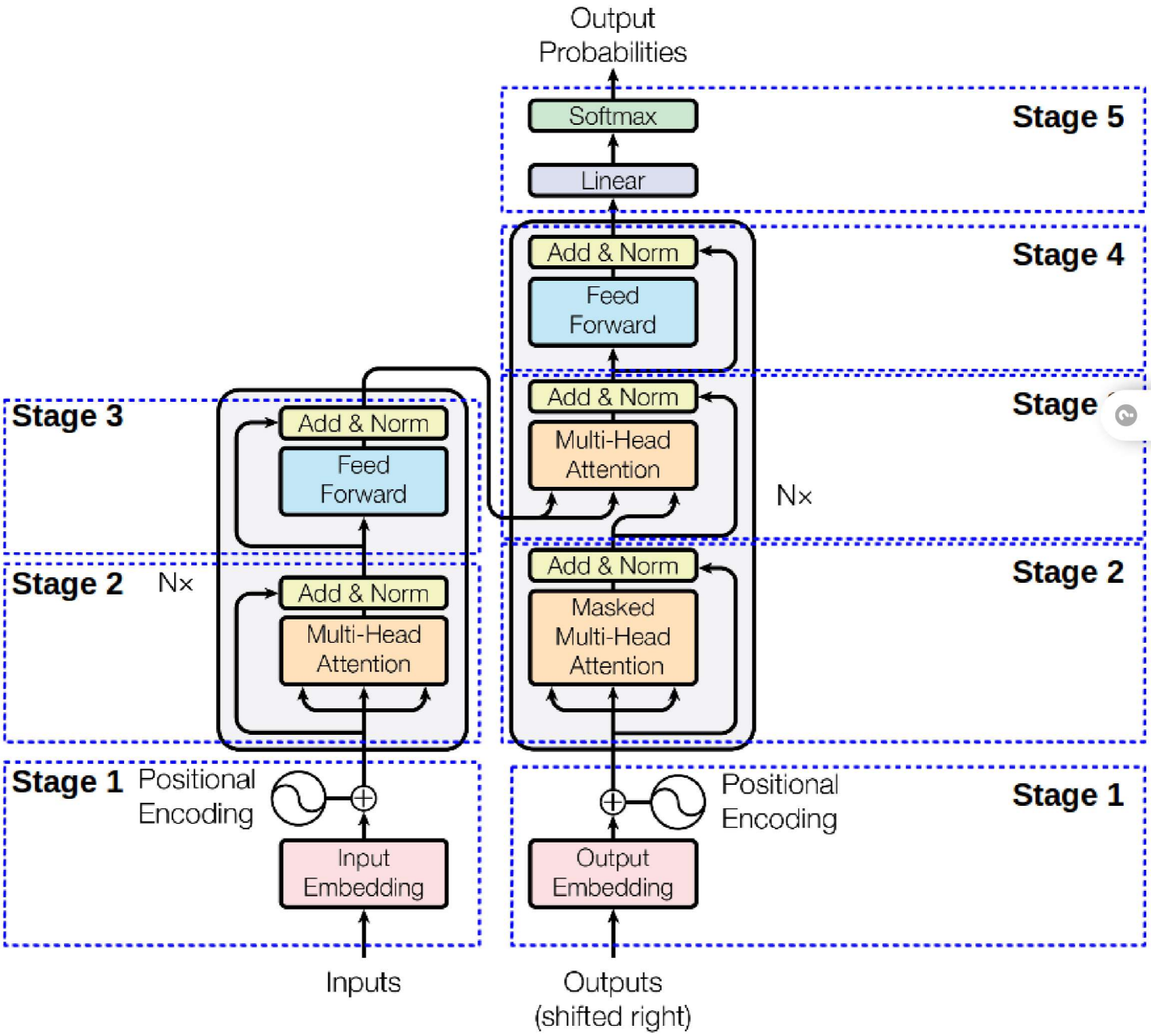
- **BERT** (https://arxiv.org/abs/1810.04805) - Bidirectional Encoder Representations from Transformers
- **GPT** (https://arxiv.org/abs/2005.14165) - Generative Pre-Trained Transformers V1-3
- **RoBERTa** (https://arxiv.org/abs/1907.11692) - Robustly Optimized BERT Pre-Training, and
- **T5** (https://arxiv.org/abs/1910.10683) - Text-to-Text Transfer Transformer.



[Transformer Models: An Introduction and Catalog (Feb 12, 2023)](https://arxiv.org/pdf/2302.07730.pdf)



The Transformer



We will decode all the blocks in the next session. Today, we will understand the fundamentals.

The profound impact of Transformer models has become more clear with their scalability to very large capacity models (GPT3). The latest Google's mixture-of-experts **Switch transformer (https://arxiv.org/pdf/2101.03961.pdf)** scales up to a whopping 1.6 Trillion parameters!

But what *can* transformers do?

1. feature extraction (get the vector representation of a text)
2. named entity representation
3. fill in the blanks
4. question-answering
5. sentiment-analysis
6. summarization
7. text-generation
8. translation
9. zero-shot-classification (classifying texts that haven't been labeled)

"Attention is all you need" is the first paper that introduced Transformers properly, but then immediately everybody jumped on **pre-trained** transformer architectures.

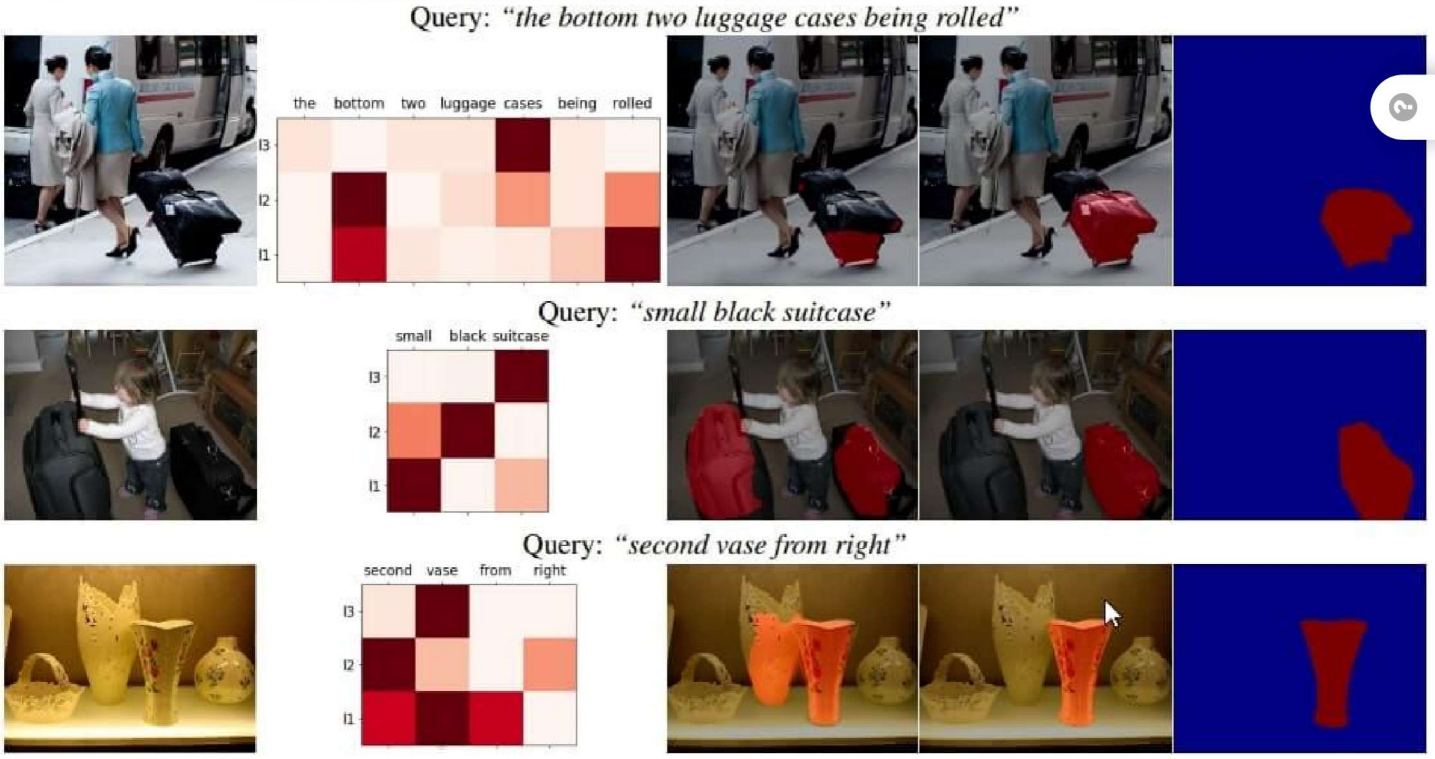
To understand Transformers fully, we need a really good understanding of many components:

1. Embedding Layer and addition of Positional Embeddings
2. Self Attention
3. Multi-Head Attention
4. Residual Connections and Normalization Process

5. Feed-forward layer and activation
6. Masking and why do we need it
7. Multi-head Attention in the Decoder
8. Final SoftMax layer for prediction

Visual data follows spatial and temporal structures, so it was challenging to apply transformers to vision. But they have been successfully applied to:

- Image Recognition - 1 (https://arxiv.org/abs/2010.11929) , 2 (https://arxiv.org/abs/2012.12877)
- Object Detection - 1 (https://arxiv.org/abs/2005.12872) , 2 (https://arxiv.org/abs/2010.04159)
- Segmentation - 1 (https://arxiv.org/abs/1904.04745)



- Image Super-Resolution - 1 (https://arxiv.org/abs/2006.04139)

- Video Understanding - 1 (https://arxiv.org/abs/1904.01766) , 2 (https://arxiv.org/abs/1812.02707)
- Image Generation - 1 (https://arxiv.org/abs/2012.00364)
- Test-Image Synthesis - 1 (https://openai.com/blog/dall-e/)
- Visual Question Answering - 1 (https://arxiv.org/abs/1908.07490) , 2 (https://arxiv.org/abs/1908.08530)

Method	VQA				GQA			$NLVR^2$	
	Binary	Number	Other	**Accu**	Binary	Open	**Accu**	Cons	**Accu**
Human	-	-	-	-	91.2	87.4	89.3	-	96.3
Image Only	-	-	-	-	36.1	1.74	17.8	7.40	51.9
Language Only	66.8	31.8	27.6	44.3	61.9	22.7	41.1	4.20	51.1
State-of-the-Art	85.8	53.7	60.7	70.4	76.0	40.4	57.1	12.0	53.5
LXMERT	**88.2**	**54.2**	**63.1**	**72.5**	**77.8**	**45.0**	**60.3**	**42.1**	**76.2**

- **Stable Diffusion** (https://stablediffusionweb.com/) [1 (https://arxiv.org/pdf/2112.10752.pdf)] or **Stable Diffusion**

(https://www.midjourney.com/) :

Most of these transformer architectures are based on the self-attention mechanism that learns the relationships between elements of a sequence. Before jumping on to self-attention, let's look at an overview of Vision Transformers

Vision Transformer

The vision transformer treats an input image as a sequence of patches.

How the ViT works in a nutshell:

1. Split the image into patches (16x16)
2. Flatten the patches
3. Produce lower-dimensional linear embeddings from the flattened patches
4. Add positional embeddings (so patches can retain their positional information)
5. Feed the sequence as an input to a standard transformer encoder
6. Pre-train the model with image labels (fully supervised on a huge dataset)
7. Finetune the downstream dataset for image classification.

Original Huge ViT was 0.6B parameters. On 10th Feb, Google released a **22B (https://arxiv.org/pdf/2302.05442.pdf)** parameter version!

FC layers with Position Encoding are really amazing and far better at storing context. One main reason for this is, in the FC layer, each neuron has full RF!

So to use Transformers or FC layers with images, we have two strategies.

1. Our models will have Convolution layers, that would extract everything till the object (block 4), and then we'll add transformers on top. The moment we finish our layer 4 (excluding the GAP or 1x1 layers to convert channels to #of classes), we'll have something like 512 channels.

2. We divide our images into 16x16x3 blocks, convolve them with 16x16x3xn kernels, and send them directly to the transformer architecture (most used strategy), for a fully FC strategy.

Embeddings

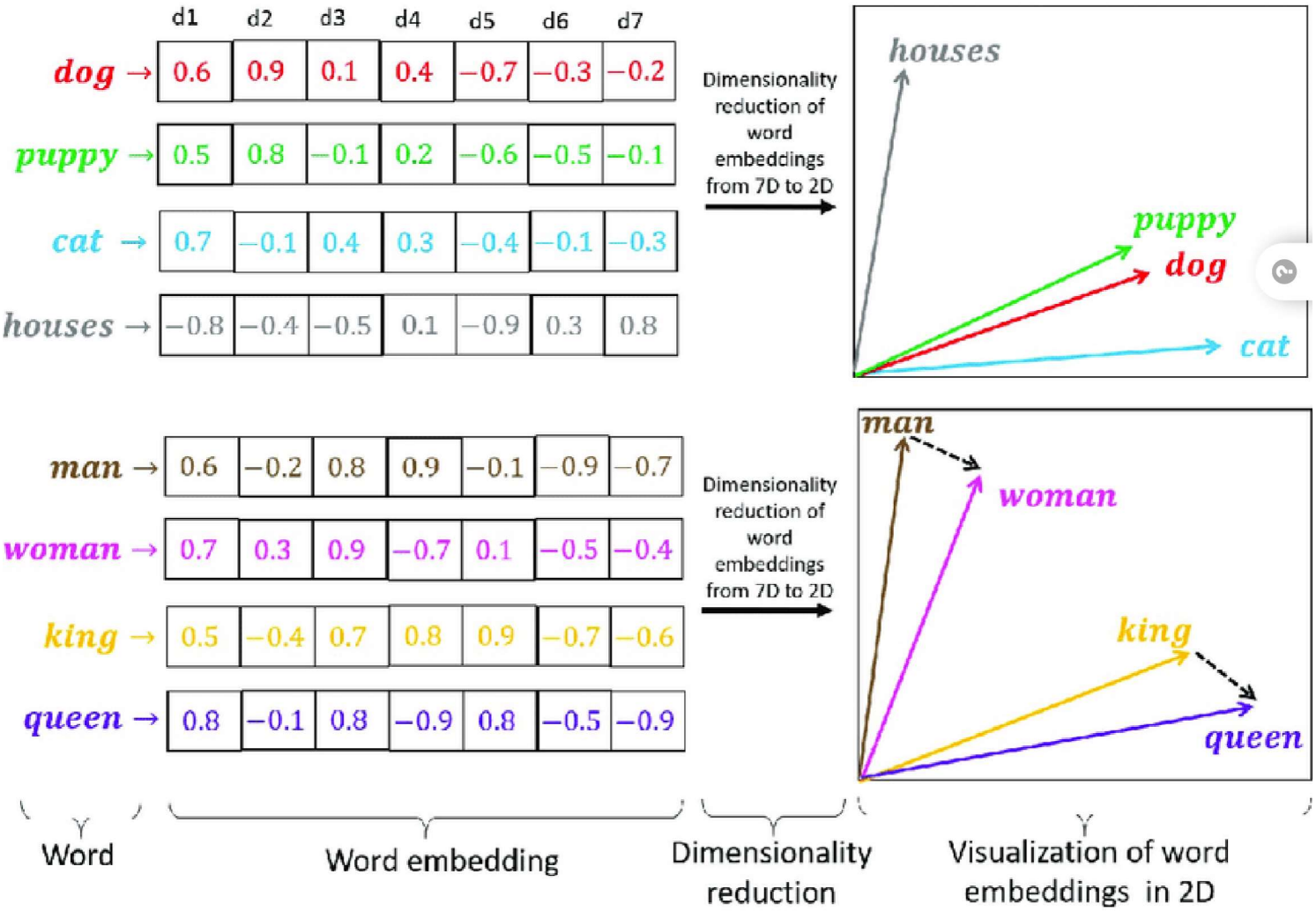
Embeddings are basically high-dimensional data represented in low-dimensional data.

So, crudely, every channel in a CNN is an embedding, but the last ones are the useful ones for us.

Remember we discussed 2 parts to our Transformer strategy for images? Well, even text has the same 2 steps. We'll calculate embedding separately, and then add a transformer block on top as the second phase.

What are word embeddings?

There used to be days when a word was just a number (scalar) in a sorted dictionary list. Then someone thought, what if we can add dimensions to each word? Imagine something like this (the top section in the image below):



But (https://indianmemetemplates.com/wp-content/uploads/2019/01/Kabhi-kabhi-lagta-hai-apun-hi-bhagwan-hai.jpg) it would be better if we let AI/ML/data decide what these

columns should mean. Suddenly differences in embeddings would also be useful!

But still, in between, there was a time when it was statistically determined, like in **GloVe (https://nlp.stanford.edu/projects/glove/)** and they went up to 300D!

The embedding dimension for GPT models is 768.

That means each word in the dictionary (say with 300k words) has 768 floating point numbers that define it!

Now, we can't just work with 300k images. We have infinite images!

So in the case of images, we train a really good CNN model (ResNet) on all possible images, work the output of the last layer for an image, and call this output the embedding of that image!

What would happen if we just take the final layer output of MNIST (embeddings) and try and cluster them using one of the **best clustering algorithms (https://nicola17.github.io/tfjs-tsne-demo/)** ?

Now that we understand embeddings, let's move to its mutation!

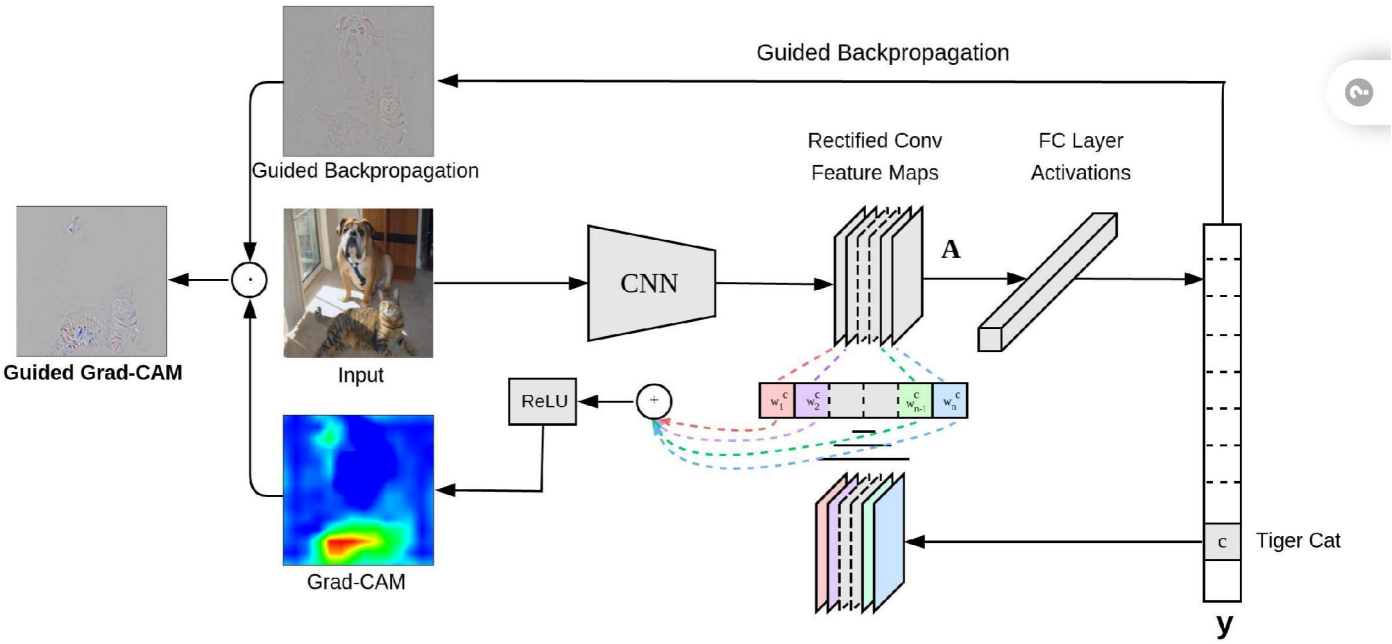
Embedding Mutation

If we want to know if a dog is a Lab, German Shepherd, Golden Retriever, or Poodle, it's somewhere there in the dog's image's embeddings.

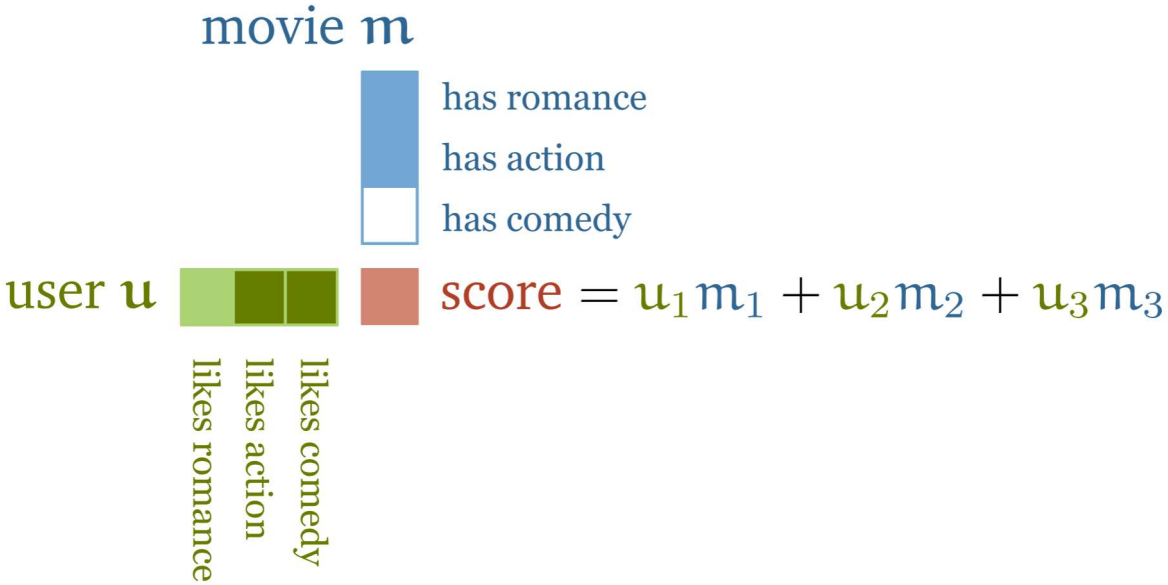
If we want to know if the vehicle is a truck, a car or a bus, it's there in its embeddings.

If we want to know if a face is smiling, angry, male, female, Asian, or old, it's there in its embeddings.

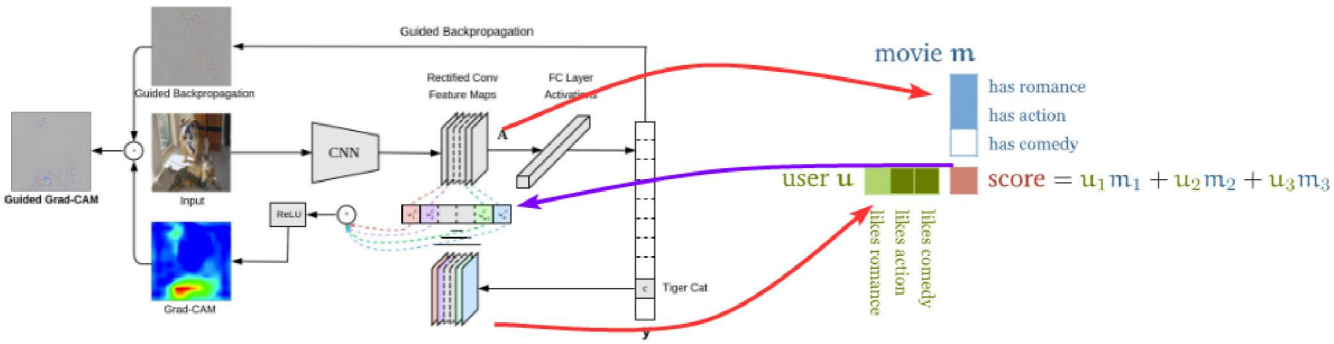
How do we extract this information, or how do we clean up the context? First, we need to realize Full Embedding is a mixture of all available contexts. So if we can somehow "weight" these dimensions/channels, we can extract just those embeddings? You already did that in GradCAM! :)



In the case of GradCAM, we know what we wanted. We got this Scaling array by the algorithm we picked.



Let's look at the similarity!



Queries: $Q = XW^Q$

Keys: $K = XW^K$

Values: $V = XW^V$

The output Z of the self-attention layer is then given by:

$$Z = softmax\left(\frac{QK^T}{\sqrt{d_q}}\right)$$

To re-emphasize again, there are 3 matrices in this sentence's solution.

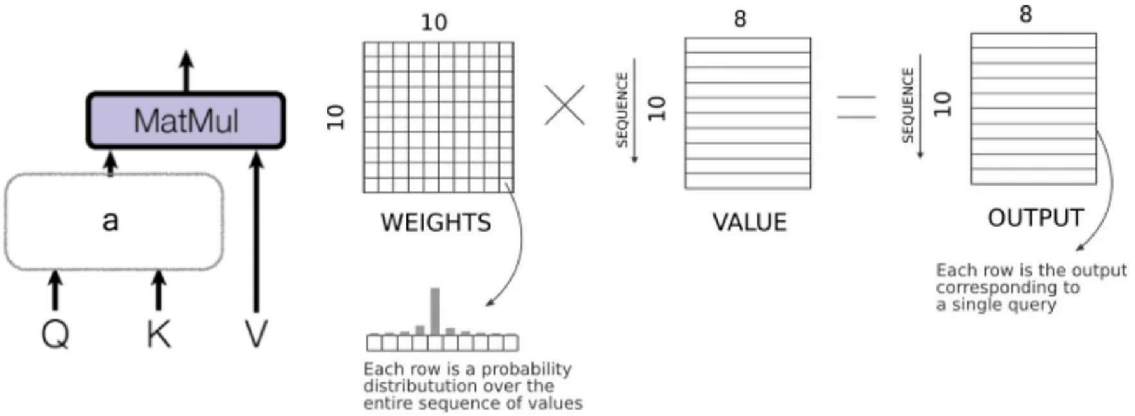
First, a Query Matrix that "reduces my dimensions"

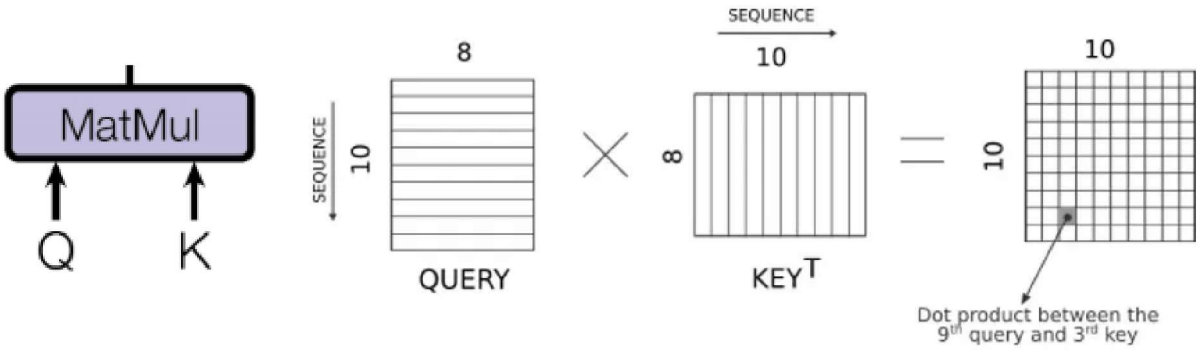
Second, a Key matrix that "calculates my cross relationships"

K*Q gives me my value Weights

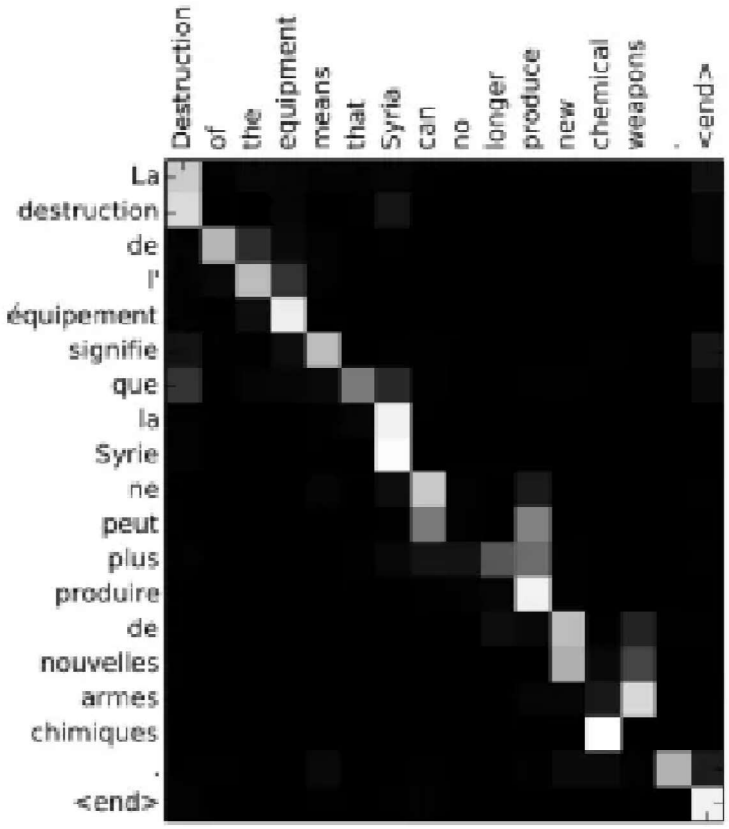
Third, the Value matrix gives me new values to be added to me to make me a "new me".

In its pure mathematical form, this is how it will look like:





The last matrix (https://medium.com/@b.terryjack/deep-learning-the-transformer-9ae5e9c5a190) on top is called the attention matrix (we'll be adding softMax to it soon) and can be visualized as:



Each word would go through a linear layer to become a query, and another one to become a key. We apply matrix multiplication between them to get the relationship (attention matrix). For 1 single work, this matrix may look like this:

0.0533 0.0398 **0.2068** 0.0715 0.0010 0.0765 0.0713 **0.2947** 0.0638 0.1207

after SM. The highest correlation is of course with itself.

Then we take all the words and convert them into "Values" that we would like to use. Again a contextual extraction. Finally, multiply the above values by these values and then add them to "my" new definition.

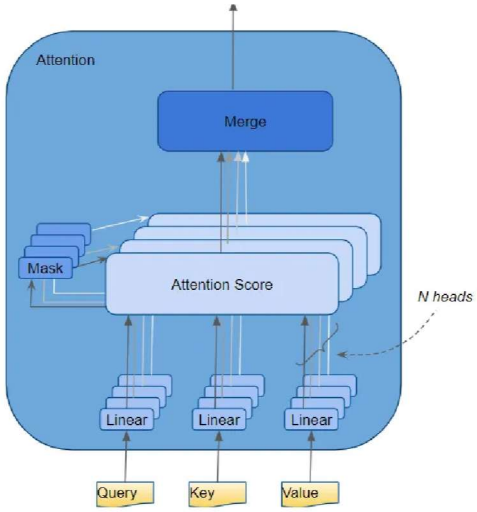
In the case of the text, remember our example "I went to the bank to withdraw money, that was next to a banking road and left to the bank of a river". In this case for the word bank, our matrix would learn from the words next to it what to clean up. **This is sort of knowing who am I based on the words that are next to me. Or self-realization or**

SELF ATTENTION!

Then there are translation tasks, where while **decoding** the translated text I need to translate based on some other context, that might be coming in from an instruction, th was **encoded**. This kind of attention is called **ENCODER-DECODER ATTENTION OR CROSS ATTENTION**.

CHANNELS

Now we are also going to leverage the concept of channels because that allows for parallelization and kernel-type specialization, so the attention algorithms will have multiple channels or heads (we don't call these channels because that will void someone's glory of introducing complexity and make things easier to understand)



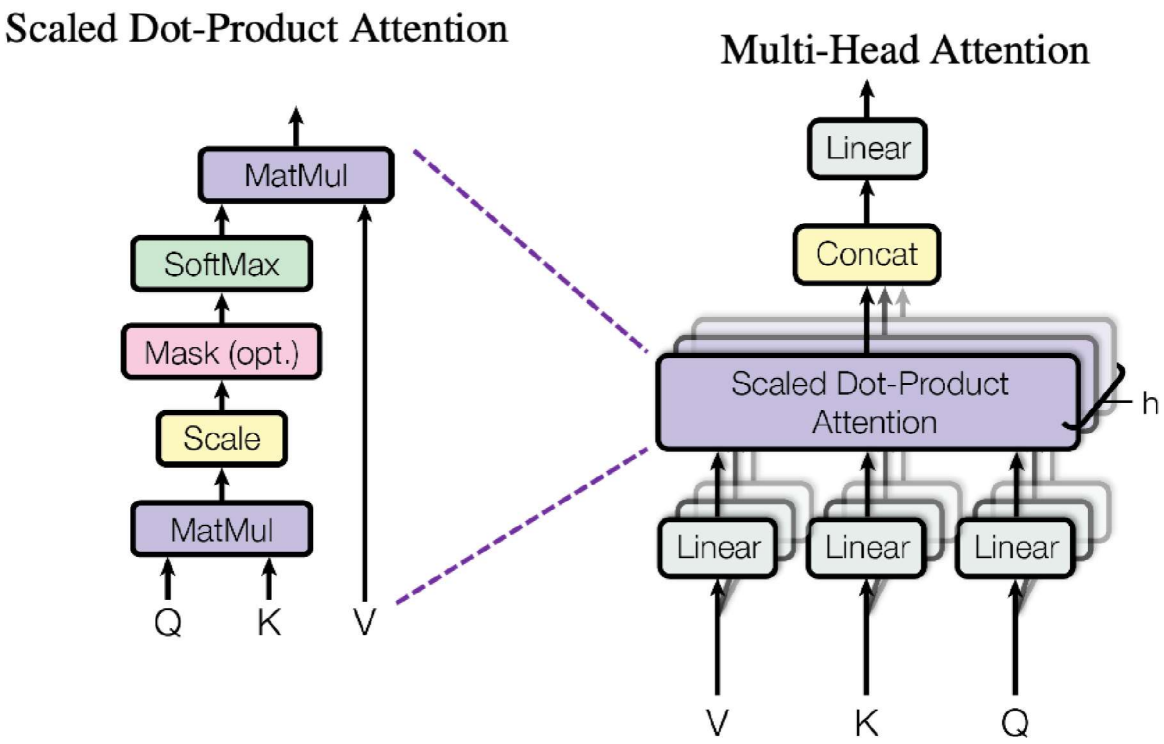
Foundation

There exist two key ideas that have contributed to the development of transformer models.

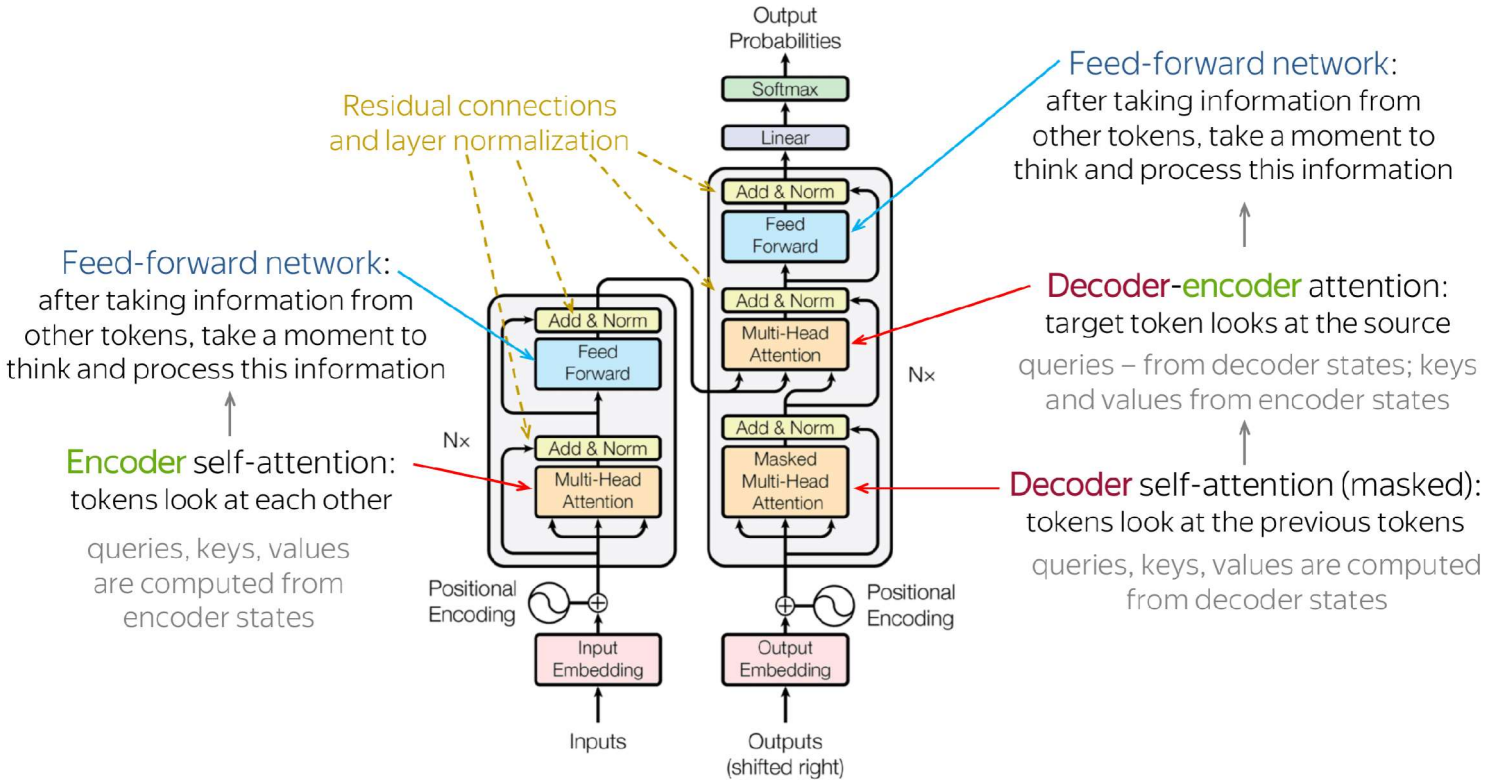
(a) The first one is self-attention, which allows capturing 'long-term' information and dependencies between sequence elements.

(b). The second key idea is that of pre-training on a large (un) labeled corpus in a (un)supervised manner, and subsequently fine-tuning to the target task with a small labeled dataset. Let's look at these background foundational concepts to better understand the forthcoming transformer-based models used in the computer vision domain.

Attention



Overall network (NOT USED)



Queries: $Q = XW^Q$

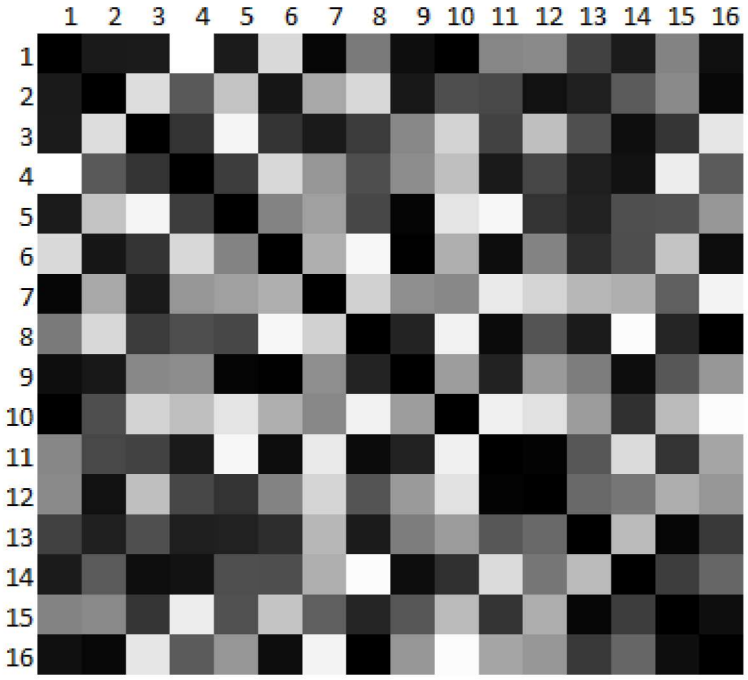
Keys: $K = XW^K$

Values: $V = XW^V$

The output Z of the self-attention layer is then given by:

$$Z = softmax\left(\frac{QK^T}{\sqrt{d_q}}\right)$$

For a given entity (16x16 block) in the sequence, the self-attention basically computed the dot product of the query with **all keys**, which is then normalized using the softmax operator to get the attention scores.



Each entity then becomes the weighted sum of all entities in the sequence, where weights are given by the attention scores.

(Un)Supervised Pre-training

The self-attention-based Transformer model generally operates in a two-stage training mechanism.

First, pre-training is performed on a large-scale dataset (and sometimes a combination of several available datasets) in either a supervised or unsupervised manner.

Later, the pre-trained weights are adapted to the downstream tasks using small-mid scale datasets. Example downstream tasks include image classification, object detection, zero-

shot learning, question-answering, and action recognition.

ViT experiences an absolute 13% drop in accuracy on ImageNet when not pre-trained on other datasets (like JFT).

Since acquiring manual labels for a massive scale is cumbersome, self-supervised learning has been very effective. As nicely summarised by Y. LeCun, the basic idea of SSL is to *fill in the blanks*, i.e. try to predict the occluded data in images, future, or past frames in temporal video sequences, or color the grayscale image, etc.

So, in the SSL-based pretraining stage, a model is trained to learn a meaningful task.

An image is worth 16x16 words

This paper presented the first implementation that completely removed convolutions.

If we were to actually do a default implementation of transformers on the images, we would look at attention units calculated at an exponential scale!

For calculating attention for each pixel in 224x224 images, we would need 50176 attention values! One alternative would be to calculate "local attention", i.e. calculating attention for local regions only.

In this paper, Google engineers figured out how to achieve global attention without getting into an exponential trap. Their idea? Patches!

They divide the image into multiple 16x16 patches. A 64x64x3 image would be divided into 16 patches. Each patch is now sent to a Conv2D layer (but it's not going to be a convolution, read further), that has the kernel size = patch_size, input channel = 3, output channel = 1, and stride = patch size. i.e.

16x16x3 | 16x16x3x1 (this is an FC layer).

Then these 16x16x1 patches are flattened, transposed, and then added with class_token (a weight initialized with 0, but allowed to be trained, we'll discuss more on this in the next session).

Then we add Position Embeddings, which are again 16+1 weights that are 0 initialized but allowed to be calculated by the backpropagation.

The output of this projection is called **patch embedding**.

These patch embeddings are then sent to the encoder.

In the encoder, these patch embeddings would be sent to a block where they are normalized, sent to MHA, added with residual connection, normalized again, send to MLP, and added with residual again. There would be 6 such blocks.

Before jumping to transformers in more detail (next session), let's look at PyTorch's **implementation**

(https://brsoff.github.io/tutorials/intermediate/spatial_transformer_tutorial.html) of Spatial Transformer Networks.

Let's take a look at **heavily encapsulated ViT code** (https://colab.research.google.com/drive/18PGLKnJ1QDkrYqDZQQp42sPo0JuBDqpy?usp=sharing) for MNIST

Assignment

No Assignment, you're getting one more week to finish off Assignment 15th!

VIDEO

STUDIO

Google Meet Version

16 GM