Computer Vision and Deep Learning

Transformers

Matthias Fulde

WS 2023/24

Transformer

- Network architecture originally developed for natural language processing tasks
  - But now also widely adapted to other domains like computer vision
  - State of the art in many tasks
- Designed to process sequential data in parallel unlike recurrent neural networks
- Computation is based on attention mechanism
  - Provides context for each element in a sequence
  - Easier to learn global relationships
- Architecture behind large language models like BERT or GPT

Attention Is All You Need, Vaswani et al., 2017

Benchmarks

- Machine translation on WMT2014 English-German

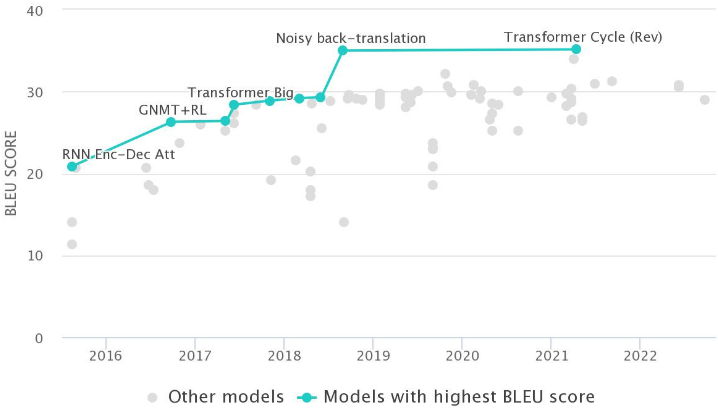


Figure from https://paperswithcode.com/sota/machine-translation-on-wmt2014-english-german

Benchmarks

- Question answering on SQuAD1.1

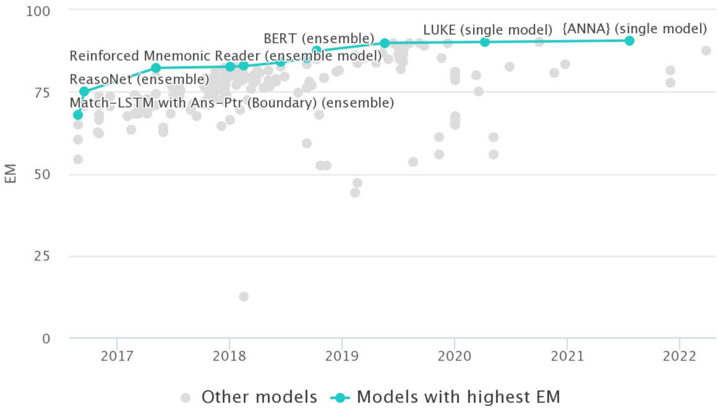


Figure from https://paperswithcode.com/sota/question-answering-on-squad11

Benchmarks

- Speech recognition on LibriSpeech test-clean

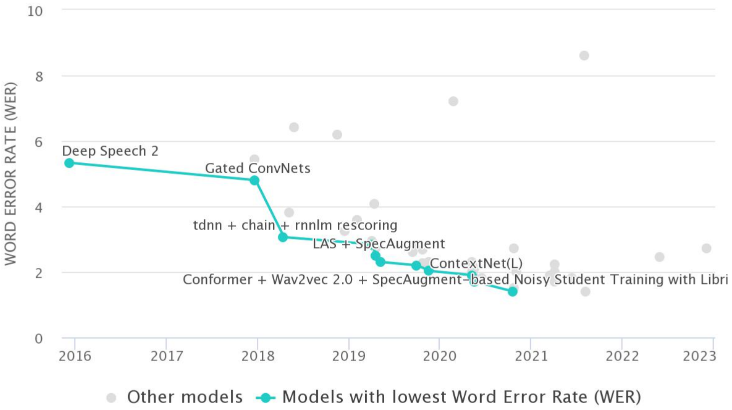


Figure from https://paperswithcode.com/sota/speech-recognition-on-librispeech-test-clean

Benchmarks

- Image classification on ImageNet

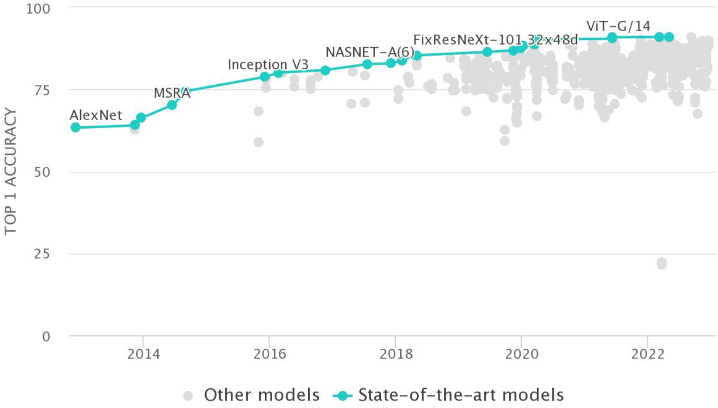


Figure from https://paperswithcode.com/sota/image-classification-on-imagenet

Benchmarks

- Object detection on COCO

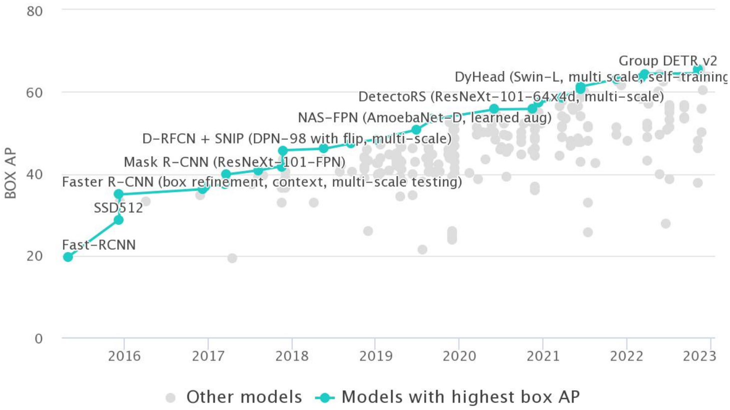


Figure from https://paperswithcode.com/sota/object-detection-on-coco

Encoder-Decoder Architecture

- Original transformer is composed of **encoder** and **decoder** networks

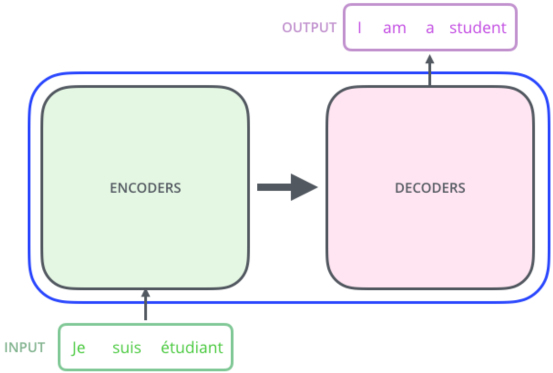


Figure from The Illustrated Transformer, Jay Alammar

Network Components

- Encoder and decoder networks are constructed as stacks of identical blocks

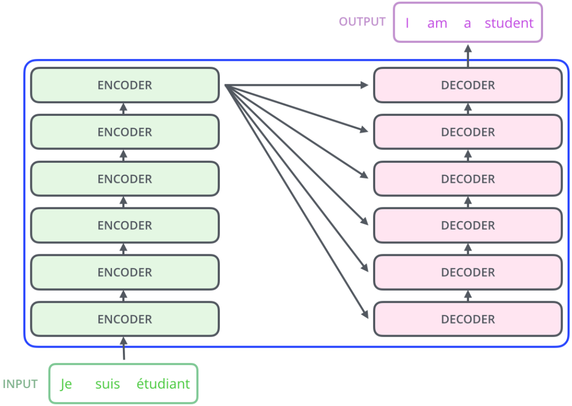


Figure from The Illustrated Transformer, Jay Alammar

Sublayers

- Each block consists of a **self-attention** sublayer and a small feed-forward network
- Decoder blocks also have an additional **cross-attention** sublayer in between

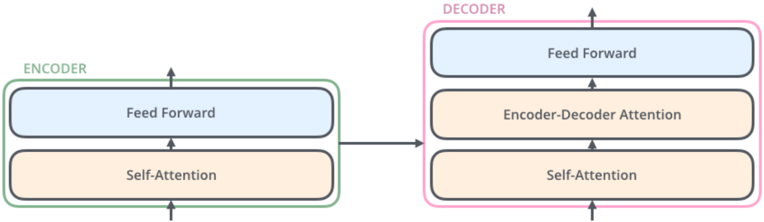


Figure from The Illustrated Transformer, Jay Alammar

Self-Attention

- Mechanism analogous to cognitive attention of humans
- Put more focus on important parts of the input and less on unimportant parts

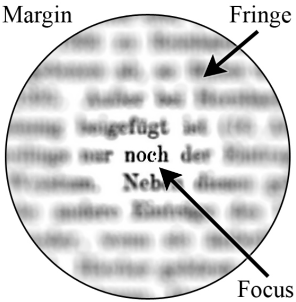


Figure from https://commons.wikimedia.org

Queries, Keys, and Values

- Transformers use attention based on feature vectors*
- Each element of the input sequence is represented as a vector $\mathbf{x}_i$
- Parameter matrices $\mathbf{W}^Q, \mathbf{W}^K$, and $\mathbf{W}^V$ are used to project each input vector
- Result is a query vector, a key vector, and a value vector

$$\mathbf{q}_i = \mathbf{W}^Q \mathbf{x}_i \qquad \mathbf{k}_i = \mathbf{W}^K \mathbf{x}_i \qquad \mathbf{v}_i = \mathbf{W}^V \mathbf{x}_i$$

  which are typically of lower dimension than the input

* For different implementations see https://en.wikipedia.org/wiki/Attention_(machine_learning)

Queries, Keys, and Values

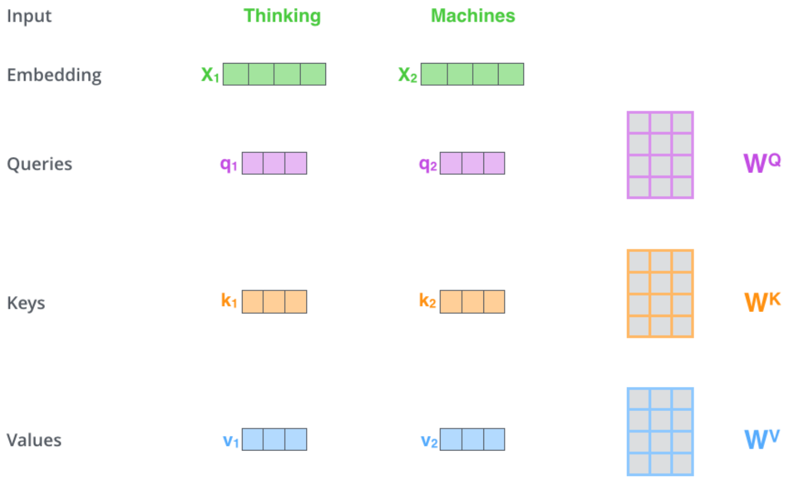


Figure from The Illustrated Transformer, Jay Alammar

Scores

- For each sequence element, its query vector is multiplied with all key vectors in the sequence to compute a score
- Idea is to find out which elements of the sequence are most important for the current element

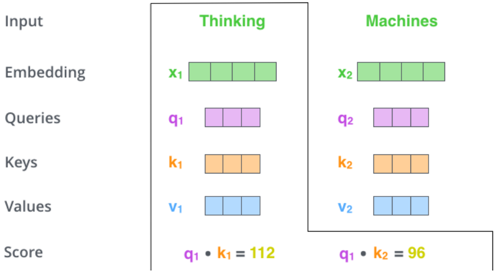


Figure from The Illustrated Transformer, Jay Alammar

Normalization

- For more stable gradients, the raw dot product scores are **scaled** with

$$\frac{1}{\sqrt{d_k}}$$

  where d_k is the dimension of the key vectors

- The softmax function

$$\text{Softmax}(\mathbf{s})_i = \frac{e^{\mathbf{s}_i}}{\sum_j e^{\mathbf{s}_j}}$$

  is used to **normalize** the scores into a probability distribution

Normalization

- Each normalized score is in the range $(0, 1)$ and the scores sum up to one
- Note that attention is computed also with respect to the element itself

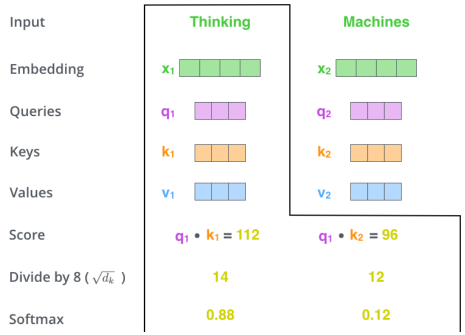


Figure from The Illustrated Transformer, Jay Alammar

Output

- For each sequence element, all value vectors are weighted with the normalized scores
- The weighted values are then summed up to generate the output for the respective sequence element

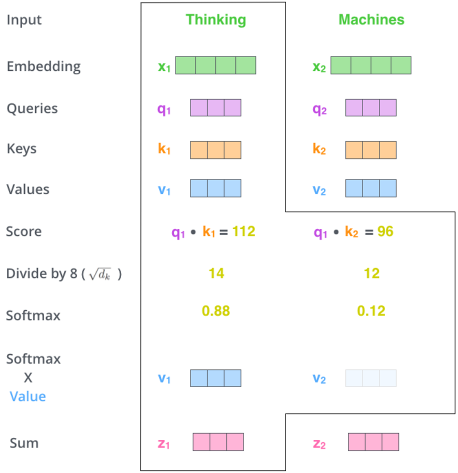


Figure from The Illustrated Transformer, Jay Alammar

Parallel Computation

- Other than in recurrent networks, sequence elements can be processed in parallel
  - Input embeddings are represented as design matrix X
  - The inputs are multiplied with the parameters to generate query, key, and value matrices Q, K, and V
  - The outputs are then computed as

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

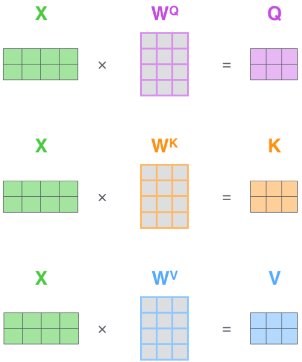


Figure from The Illustrated Transformer, Jay Alammar

Multi-head Attention

- So far we discussed only attention with a single attention head
- Problematic because we can expect that for each sequence element, the element itself has the most importance and less attention is paid to other elements
- Solution is to compute multiple attention maps using different parameter matrices, such that inputs are projected into different representational spaces
- Result is that each element can attend to multiple parts of the sequence

Different Parameters

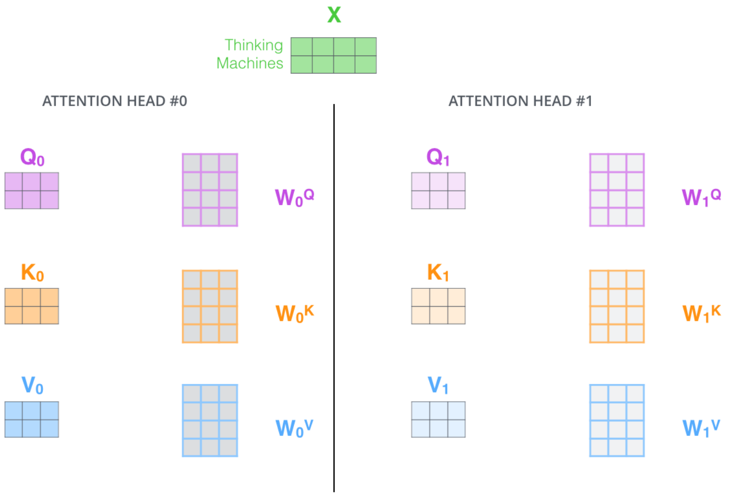


Figure from The Illustrated Transformer, Jay Alammar

Different Outputs

- Each attention head generates a different output matrix Z_h

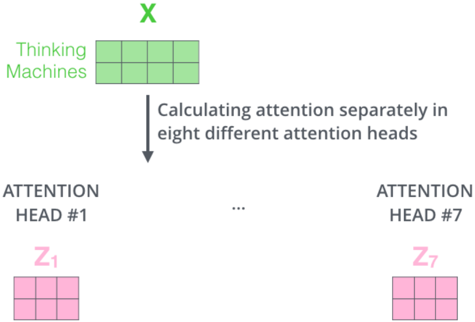


Figure from The Illustrated Transformer, Jay Alammar

Concatenation and Projection

- Another matrix W^O is used to project the concatenated outputs

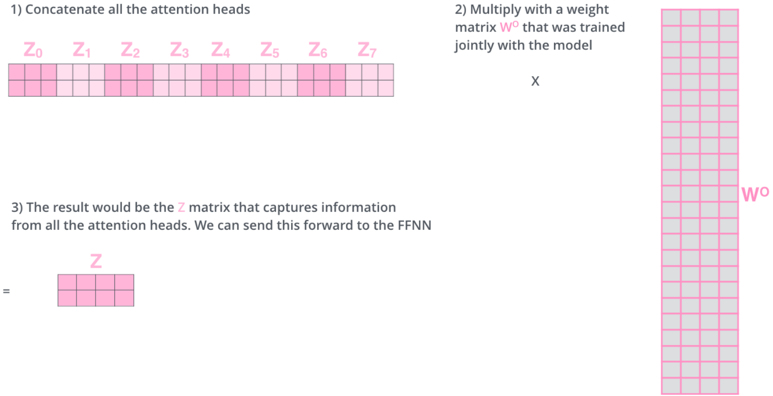


Figure from The Illustrated Transformer, Jay Alammar

Summary

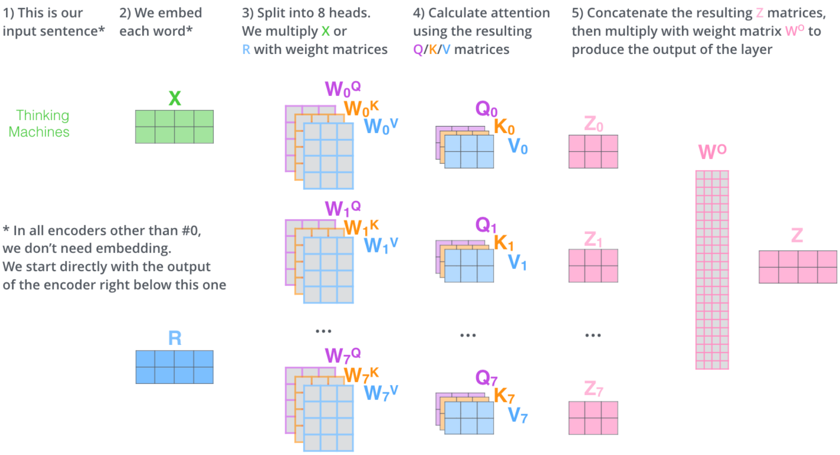


Figure from The Illustrated Transformer, Jay Alammar

Encoder Blocks

- Each encoder block is composed of a multi-head self-attention sublayer and a small two-layer feed-forward network
- There are residual connections passing both sublayers and layer normalization is applied after merging the branches

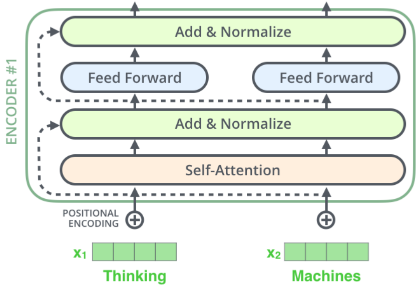


Figure from The Illustrated Transformer, Jay Alammar

Layer Normalization

- Using layer normalization in transformers means that mean and variance are computed across all features of a single input sequence

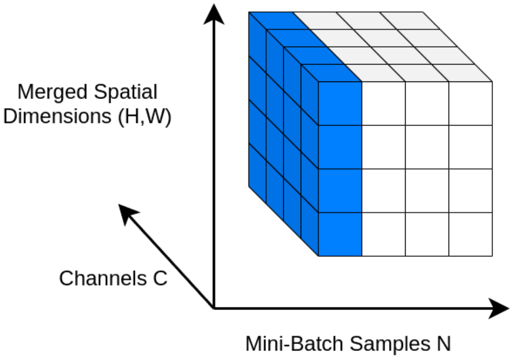


Figure from https://theaisummer.com/normalization/

Placement

- The previously described architecture is also known as **Post-LN** transformer architecture
- A more stable training with easier hyperparameter tuning can be achieved using a **Pre-LN** transformer architecture
- Here the normalization is applied before the self-attention and fully-connected sublayers

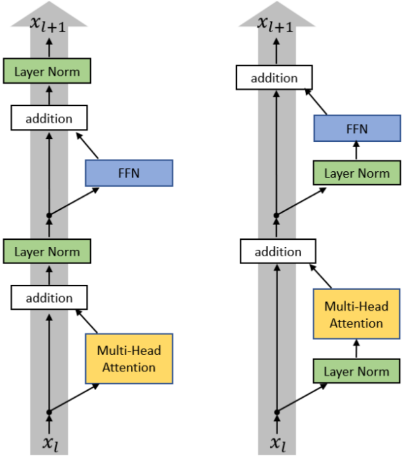


Figure from On Layer Normalization in the Transformer Architecture, Xiong et al., 2020

Decoder Blocks

- Decoder blocks work pretty much the same as encoder blocks except that they have an additional **cross-attention** sublayer between the self-attention and fully-connected sublayers
- In the cross-attention sublayer, only query matrices Q_h are computed from the current input sequence
- The key and value matrices K_h and V_h are computed from the output of the last encoder block
- Idea is that when generating a new output element, each decoder block should have access to the feature representation of the input sequence created by the encoder network

Overview

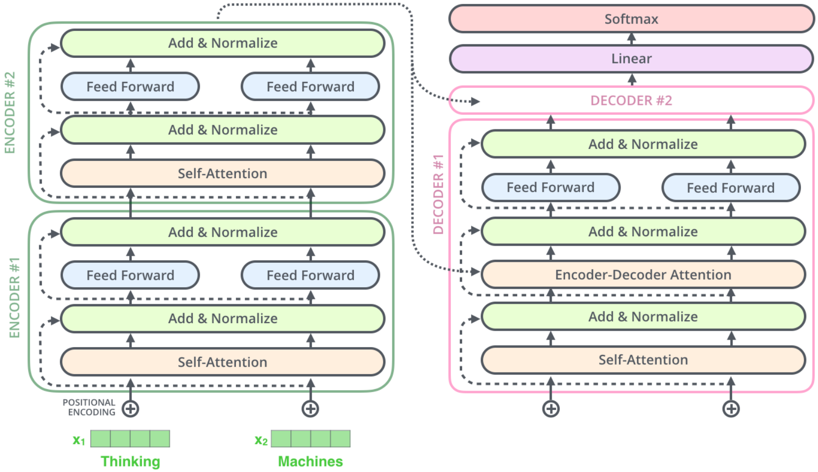
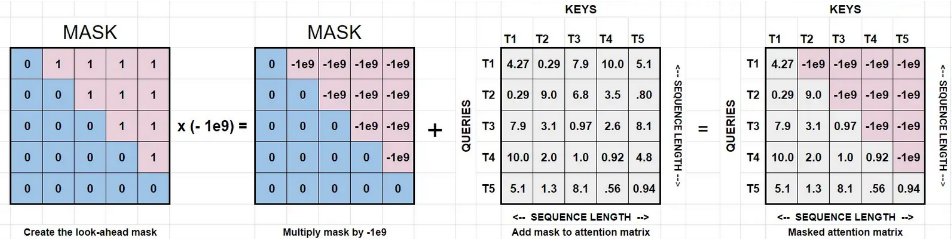


Figure from The Illustrated Transformer, Jay Alammar

Masking

- The decoder generates output elements step by step
- The first and last elements of the output sequence are special symbols denoting the start and end of the sequence
- In order to allow self-attention in the decoder blocks, the elements corresponding to not yet generated outputs are masked



Output

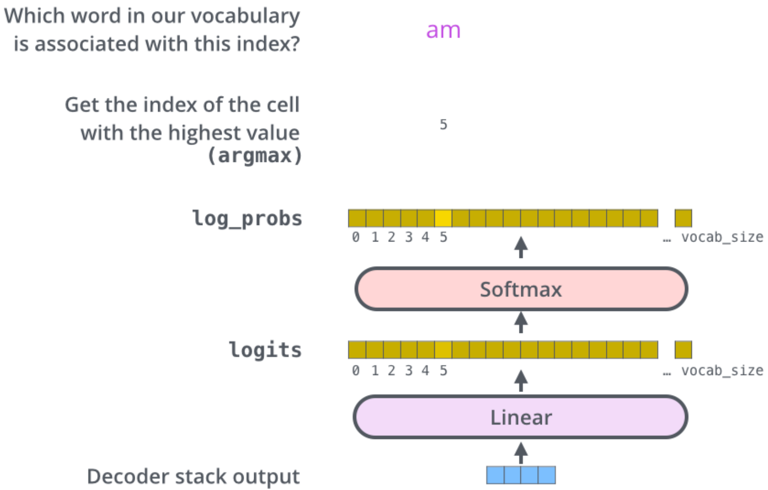
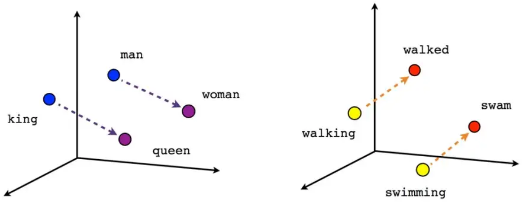


Figure from The Illustrated Transformer, Jay Alammar

Word Embedding

- Assuming natural language input, we have to translate this into a numerical representation before giving it to the transformer
- Some algorithms that can be used to perform this task are word2vec and GloVe
- Ideally, vector representations should be low dimensional while preserving contextual similarity



Efficient Estimation of Word Representations in Vector Space, Mikolov et al., 2013
GloVe: Global Vectors for Word Representation, Pennington et al., 2014
Figure from Glossary of Deep Learning: Word Embedding, Jaron Collis

Positional Encoding

- Transformer as described so far has no built-in mechanism to take into account the order of the sequence
- To address this problem, a **positional encoding** can be added to the elements of the sequence
- Position encodings can be computed using a fixed function or implemented as a learnable position bias
- Absolute or relative positions can be encoded

Fixed Function Positional Encoding

- Original transformer uses a fixed function positional encoding based on sinusoids
- The created encoding vectors have the same length as the feature vectors for the elements of the sequence, so that they can be added together

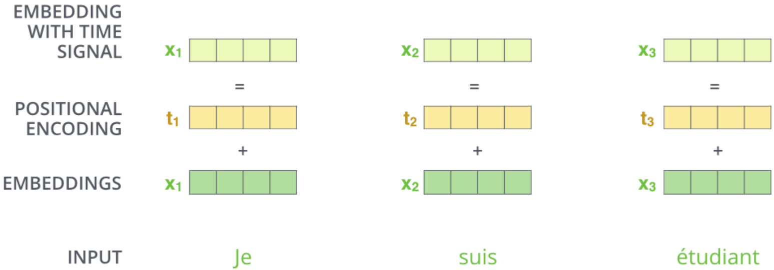


Figure from The Illustrated Transformer, Jay Alammar

Formula

- The encoding vectors contain sine and cosine functions with different frequencies in each dimension
- With pos being the position within the sequence and i being the dimension, the vector components are computed with

$$\mathsf{PE}_{pos,2i} = \sin\left({}^{pos}\!/_{10000^{2i/d_{\mathsf{model}}}}\right)$$

and

$$\mathsf{PE}_{pos,2i+1} = \cos\left({}^{pos}\!/_{10000^{2i/d_{\mathsf{model}}}}\right)$$

where d_{model} is the dimension of the feature vectors

Visualization

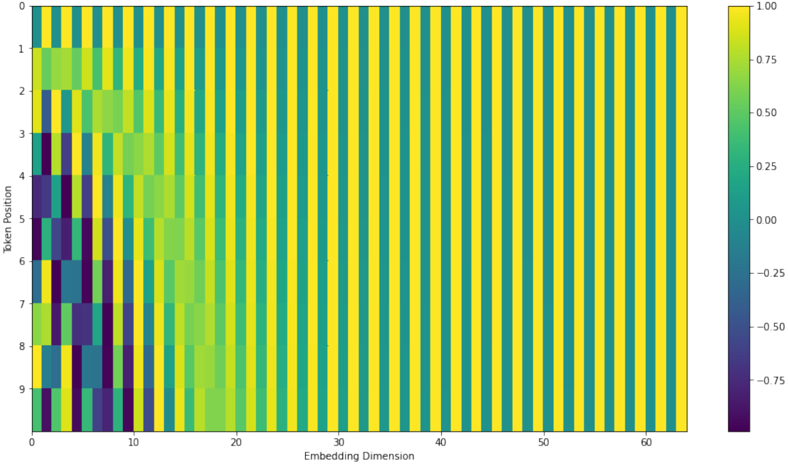


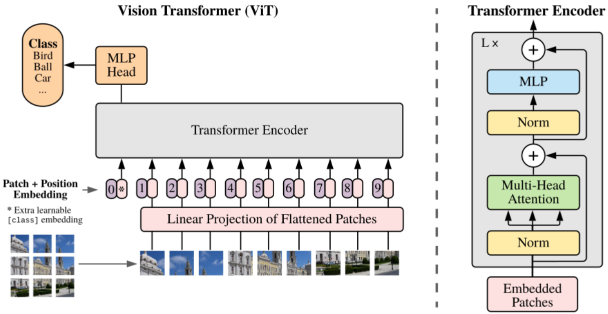
Figure from The Illustrated Transformer, Jay Alammar

Vision Transformers

- Transformers have been successfully adapted to the vision domain
- Main challenges for the application are
  - Grid structure of images instead of sequential structure of language
  - Computational complexity, since computational cost is quadratic in the number of sequence elements
  - No inductive bias for locality as in convolutional neural networks
  - Training typically requires large amount of data
- Main advantage is
  - Transformers can take into account the whole image context and are therefore more capable to learn relationships between distant parts of an image

Architecture

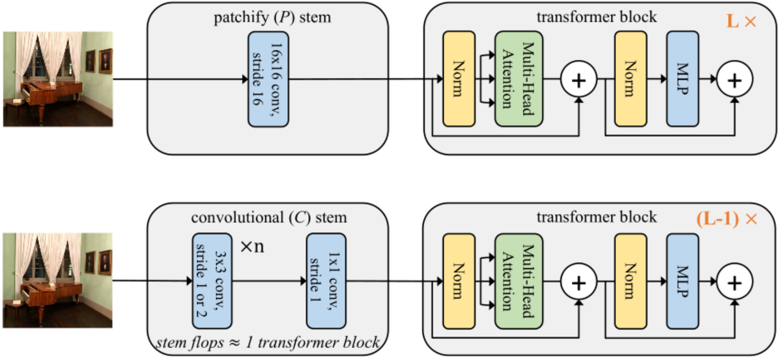
- First vision transformers developed to be close to original architecture, but only the encoder network is used
- Images are partitioned into non-overlapping patches which are converted into vectors



An image is worth 16×16 Words: Transformers for Image Recoginition at Scale, Dosovitskiy et al., 2021

Improvements

- Easier optimization and faster convergence can be achieved by using a convolutional stem to extract low-level image features



Early Convolutions Help Transformers See Better, Xiao et al., 2021

Swin Transformer

- More advanced approach to adapt transformers to the vision domain
- Main idea is to attend only within **local windows** to reduce computational cost, but to **shift** and **merge** these windows in subsequent layers in order to retain the ability to learn global dependencies

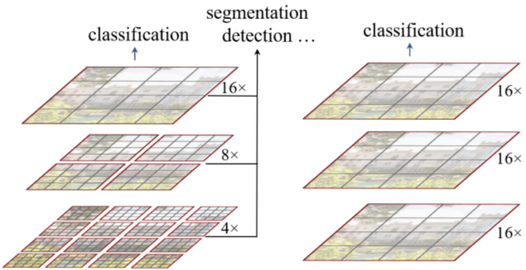


Figure from Swin Transformer: Hierarchical Vision Transformer using Shifted Windows, Liu et al., 2021

Shifted Windows

- Image is partitioned into patches as in original vision transformer
- Attention is only computed between patches belonging to the same window
- Shifting windows allows to make connections across previous window boundaries

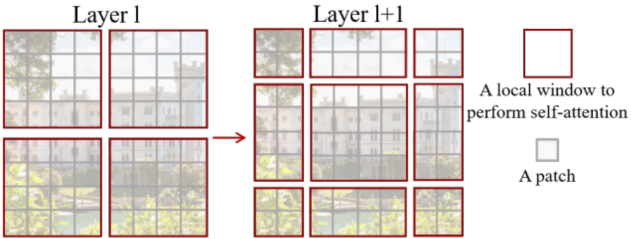


Figure from Swin Transformer: Hierarchical Vision Transformer using Shifted Windows, Liu et al., 2021

Architecture

- Proposed network architecture and view on subsequent transformer layers

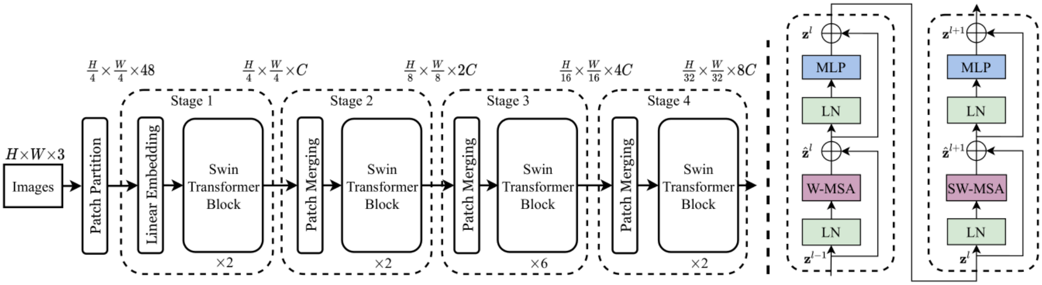


Figure from Swin Transformer: Hierarchical Vision Transformer using Shifted Windows, Liu et al., 2021