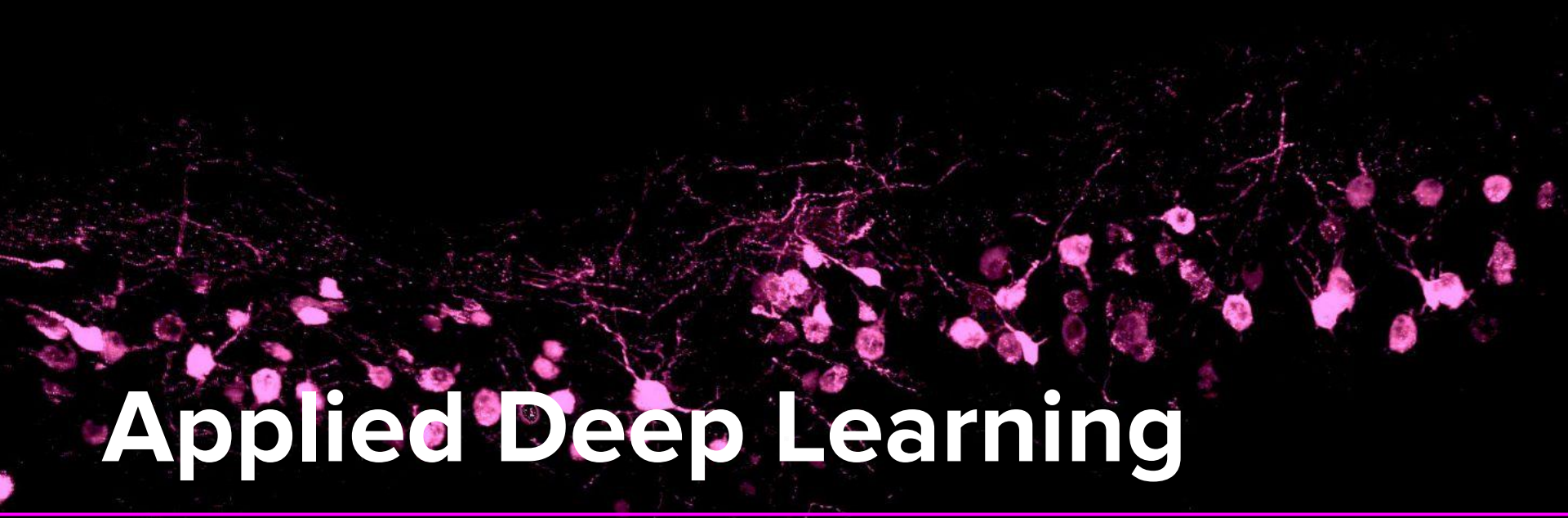


Applied Deep Learning

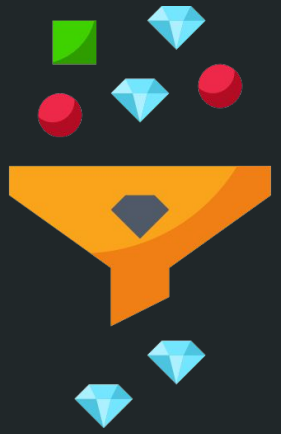
Transformers

Alexander Pacha - TU Wien

Recap

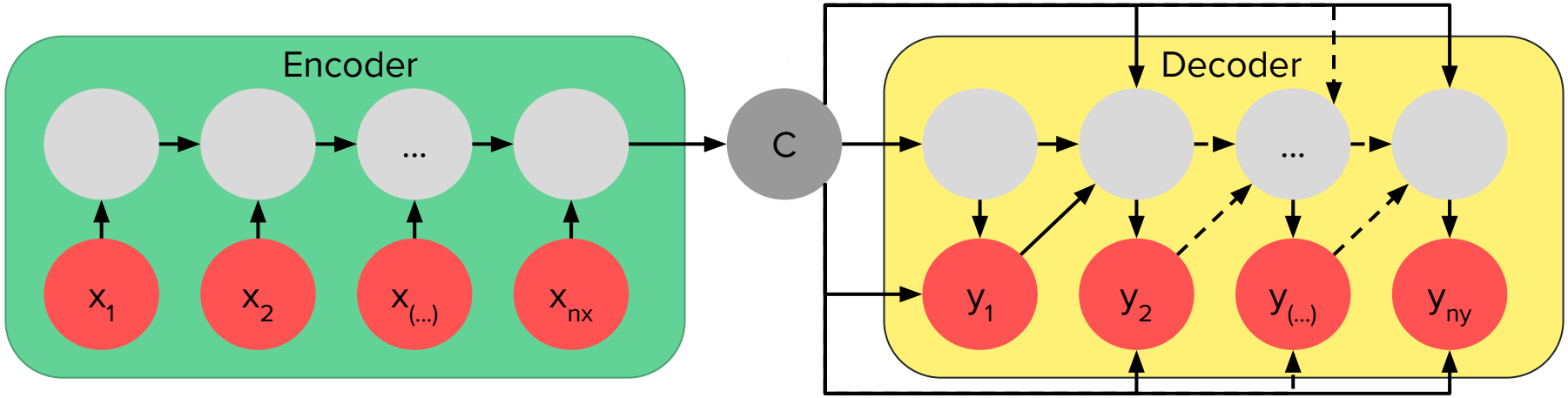
- What is the relation between the input and output of an autoencoder?
- What are the two main blocks in an autoencoder?
- What can you use an autoencoder for?
- How are variational autoencoders different from standard autoencoders?
- What does the Kullback-Leibler divergence measure?
- What are the two main blocks in a generative adversarial network?

Transformers



Sequence Modeling with Deep Neural Networks

- Popular choice: Recurrent neural networks with Encoder-Decoder architecture
- Input and output sequences can have different lengths:
  - Process input with encoder (reader) into context C
  - Decoder (writer) generates output from context C



Problem: Long-Term Memory

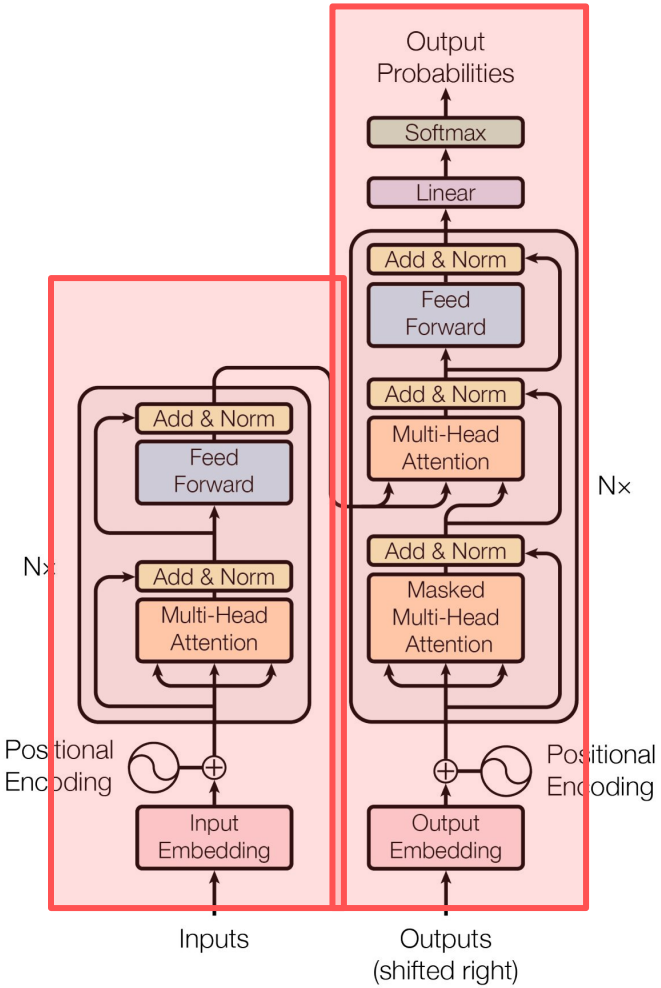
- CNNs: Fixed window
- RNNs: Short reference window
- LSTMs/GRUs: Longer reference window

“When I woke up, I saw the most incredible thing that I’ve ever witnessed. I tried to understand what was happening, but at that moment, I just stared blankly at it.”

Transformers LSTM RNN CNN

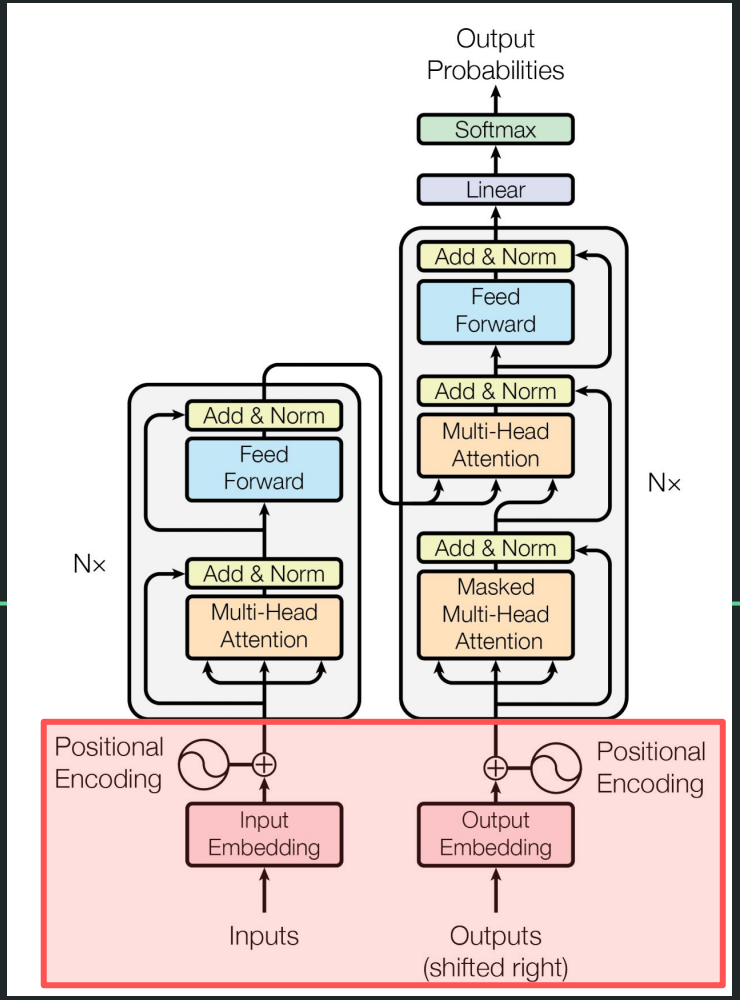
What is a Transformer?

- Architecture based on auto-regressive Encoder-Decoder architecture
- Input sequence can be processed in parallel
- Key-ingredient: Attention mechanism



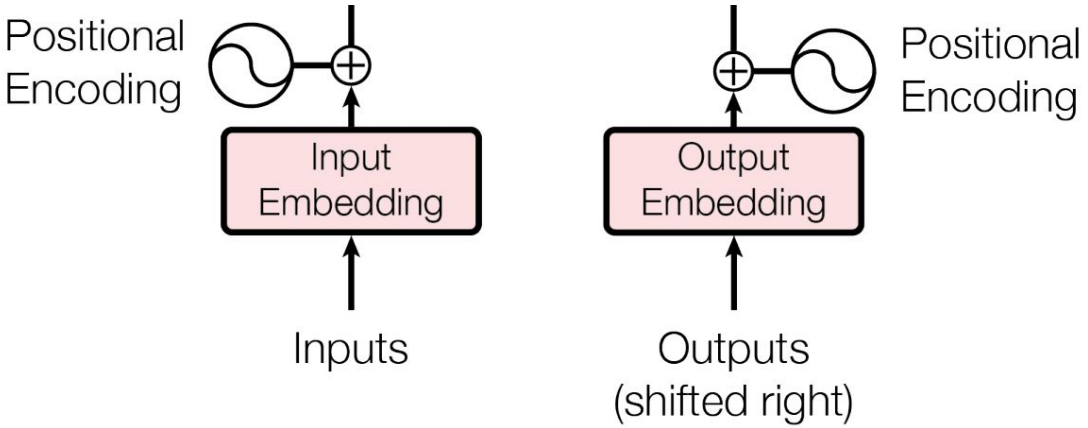
Source: [2]

Input Embedding & Positional Encoding



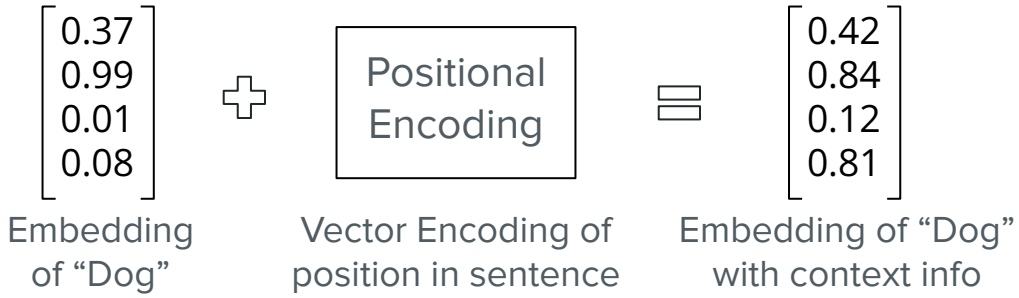
Input embedding and positional encoding

- Input embedding transforms the input into a vector
  - Words in language processing using Word2Vec
  - Image to features using CNN backbone



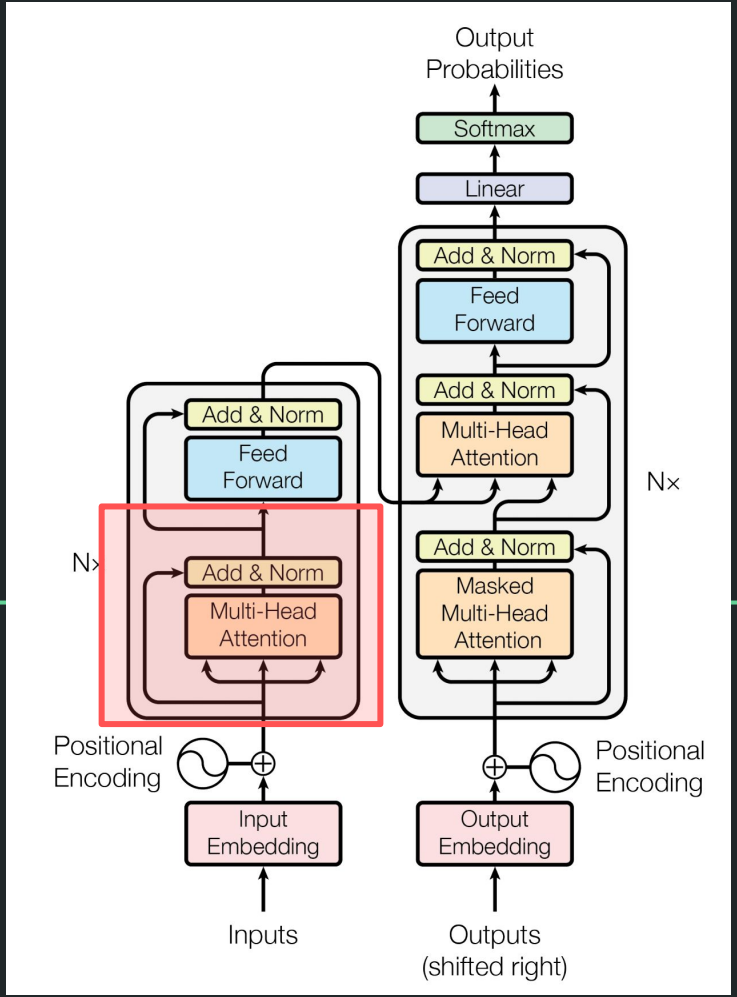
Input embedding and positional encoding

- Input embedding transforms the input into a vector
  - Words in language processing using Word2Vec
  - Image to features using CNN backbone
- Positional Encoding adds contextual information of the position within the input sequence



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

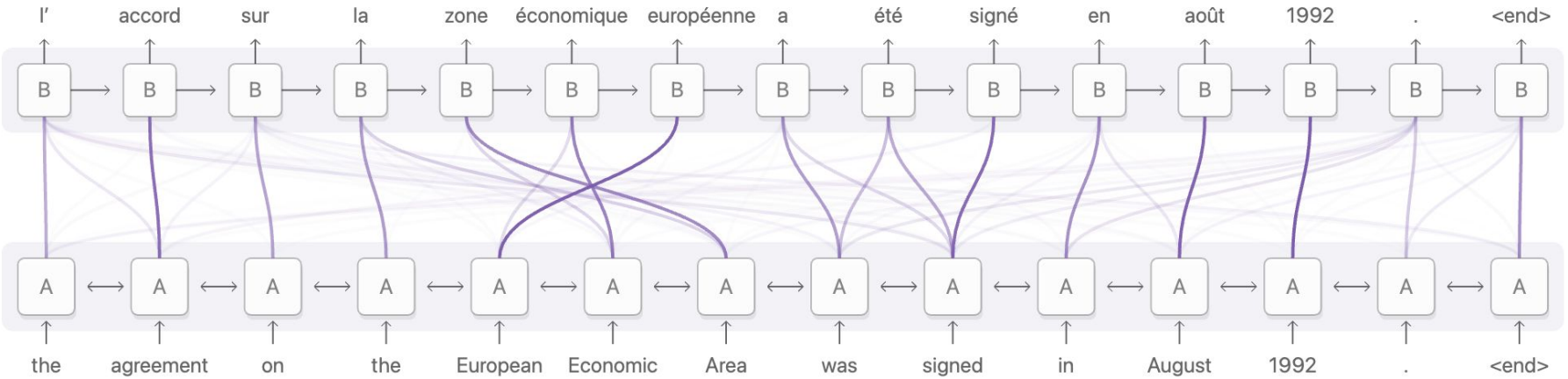
Attention


Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Add & Norm
Masked Multi-Head Attention
Nx
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

Attention and Self-Attention

What part of the input should be focused on?

- Captures contextual information between elements in the input sequence

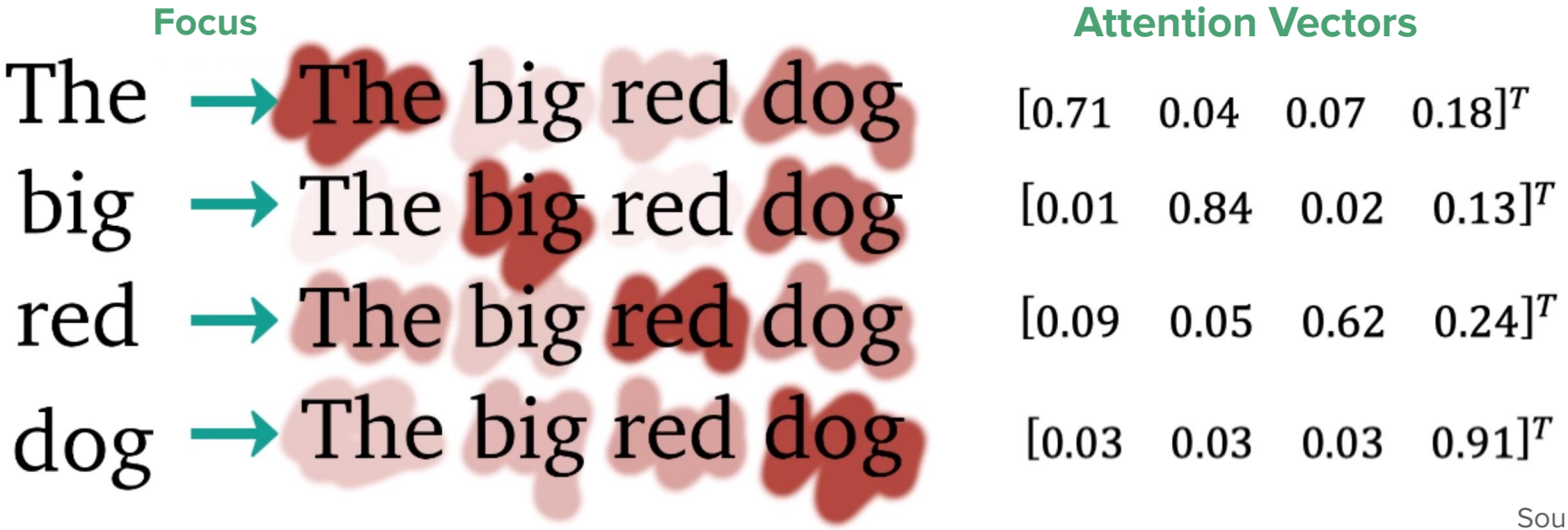


Source: [9]

Attention and Self-Attention

What part of the input should be focused on?

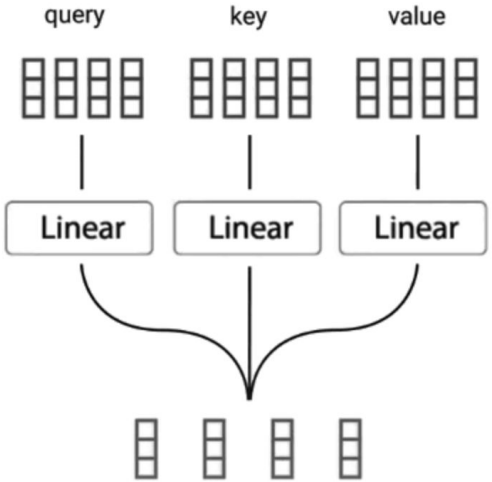
- Captures contextual information between elements in the input sequence
- Self-Attention associate each word of the input sequence to every other word

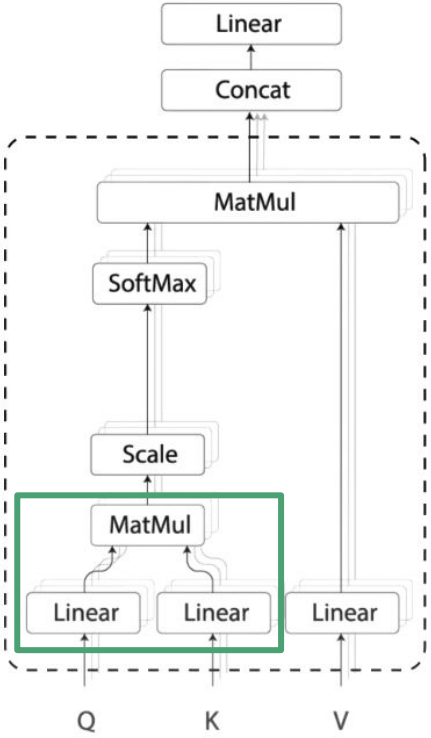


Source: [1]

Implementing Attention

- Conceptualized as a retrieval problem with a Query lookup in a set of Key-Value pairs
- Implemented by a series of matrix multiplications

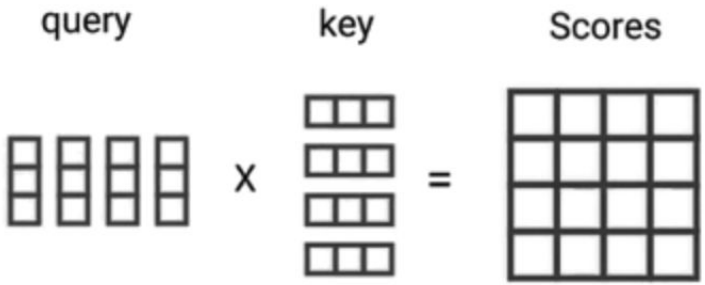


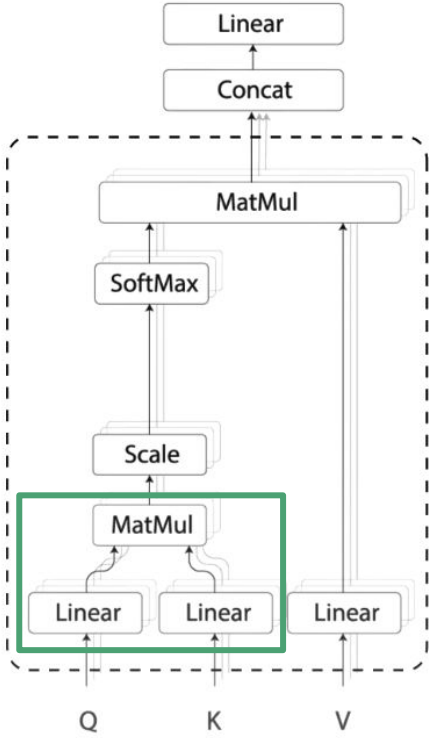


Source: [8]

Implementing Attention

- Conceptualized as a retrieval problem with a Query lookup in a set of Key-Value pairs
- Implemented by a series of matrix multiplications



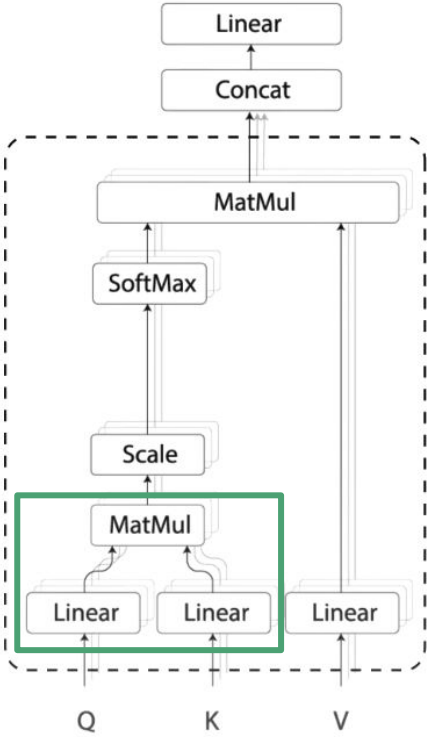


Source: [8]

Implementing Attention

- Conceptualized as a retrieval problem with a Query lookup in a set of Key-Value pairs
- Implemented by a series of matrix multiplications

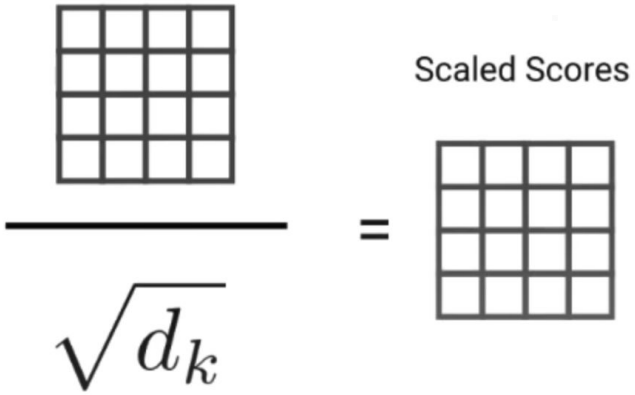
	Hi	how	are	you
Hi	98	27	10	12
how	27	89	31	67
are	10	31	91	54
you	12	67	54	92

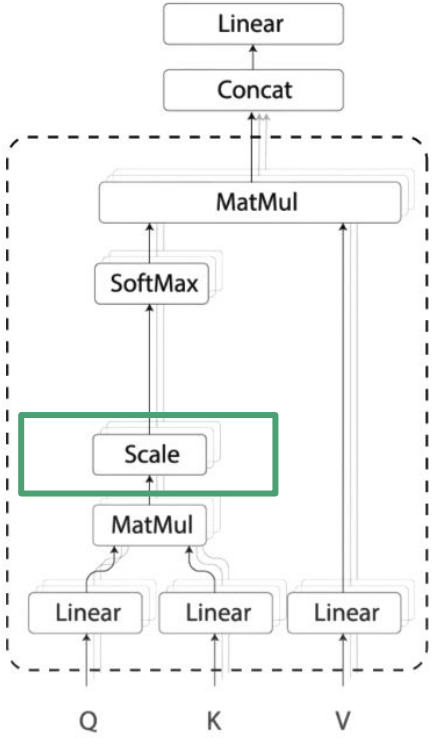


Source: [8]

Implementing Attention

- Conceptualized as a retrieval problem with a Query lookup in a set of Key-Value pairs
- Implemented by a series of matrix multiplications





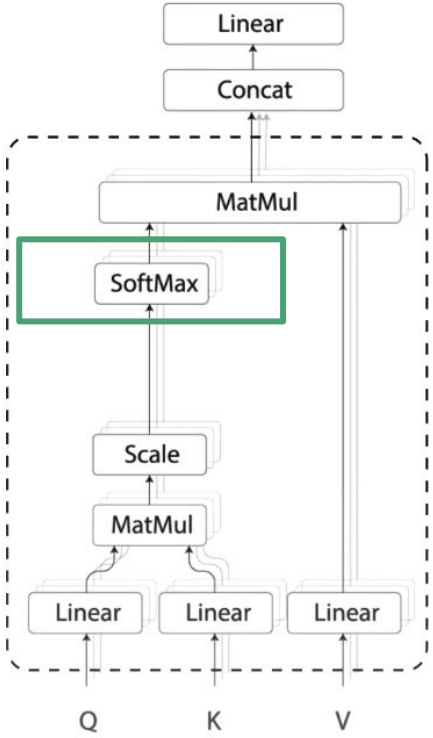
Source: [8]

Implementing Attention

- Conceptualized as a retrieval problem with a Query lookup in a set of Key-Value pairs
- Implemented by a series of matrix multiplications

Softmax(▦) =

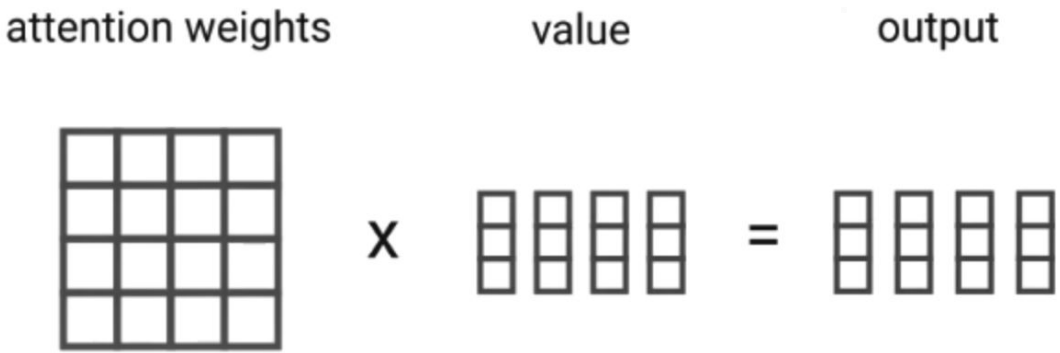
	Hi	how	are	you
Hi	0.7	0.1	0.1	0.1
how	0.1	0.6	0.2	0.1
are	0.1	0.3	0.6	0.1
you	0.1	0.3	0.3	0.3

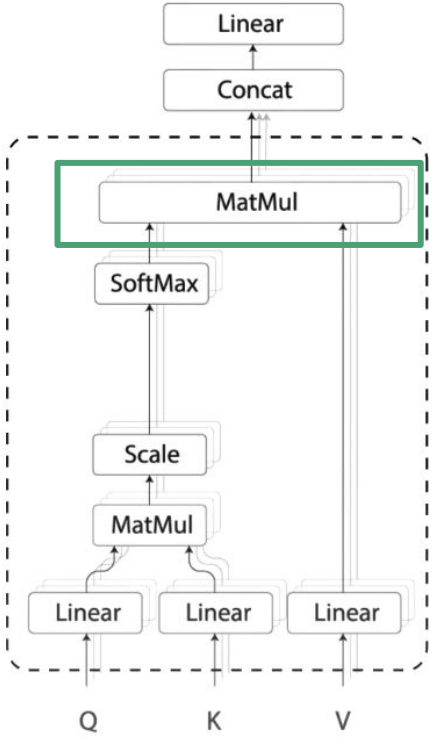


Source: [8]

Implementing Attention

- Conceptualized as a retrieval problem with a Query lookup in a set of Key-Value pairs
- Implemented by a series of matrix multiplications

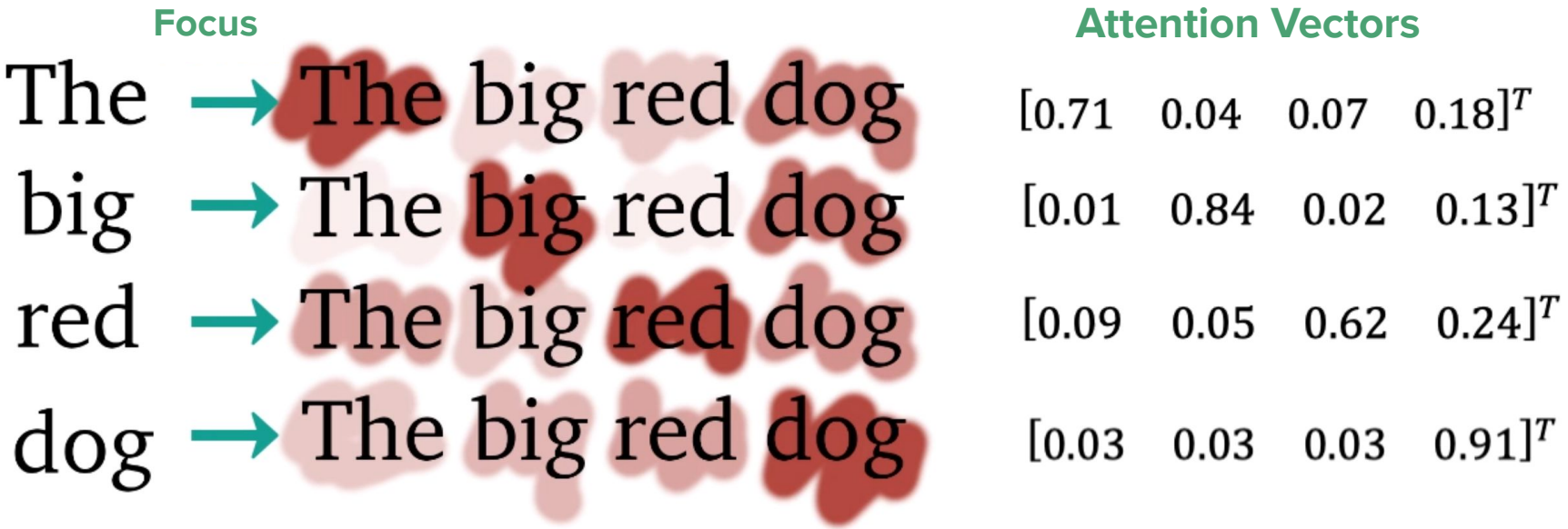




Source: [8]

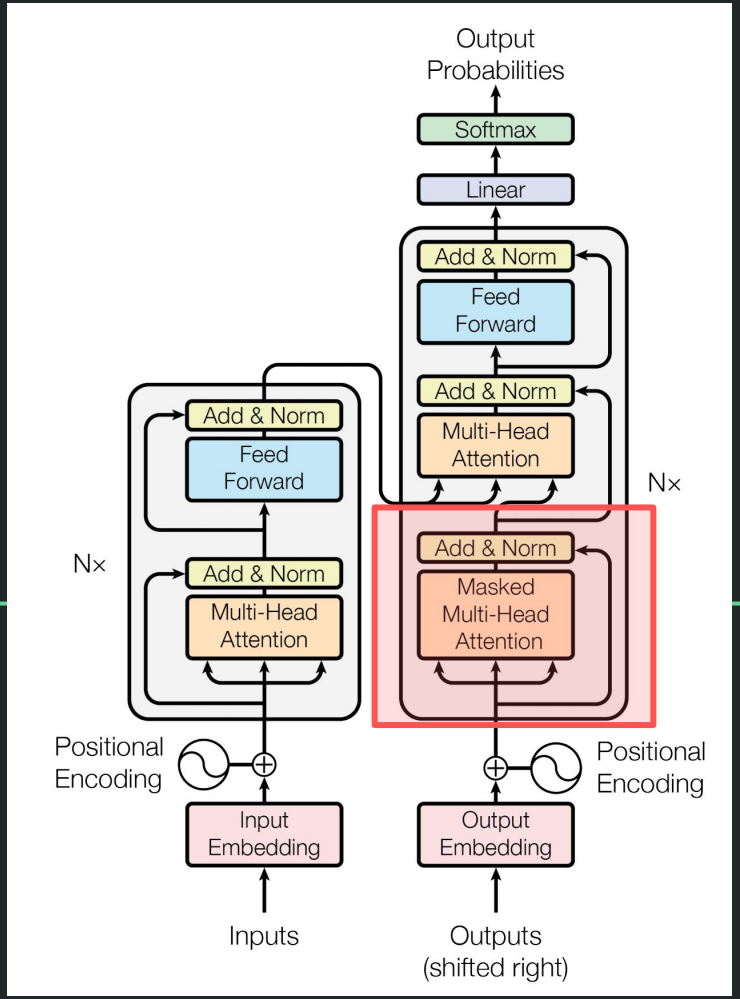
Multi-Headed Attention

- Self-attention tends to put too much weight on itself.
- Replicating the attention layer and averaging the result reduces this problem



Source: [1]

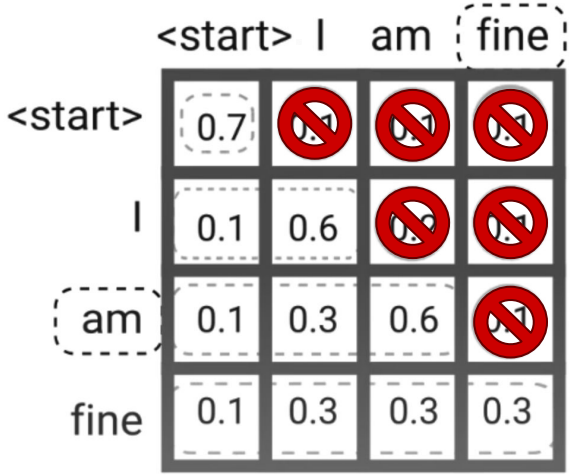
Masked Attention



Masked (Multi-headed) Attention

Attention in decoder very similar to attention in encoder, but:

- When processing the output sequence, we must only look into the past



Masked (Multi-headed) Attention

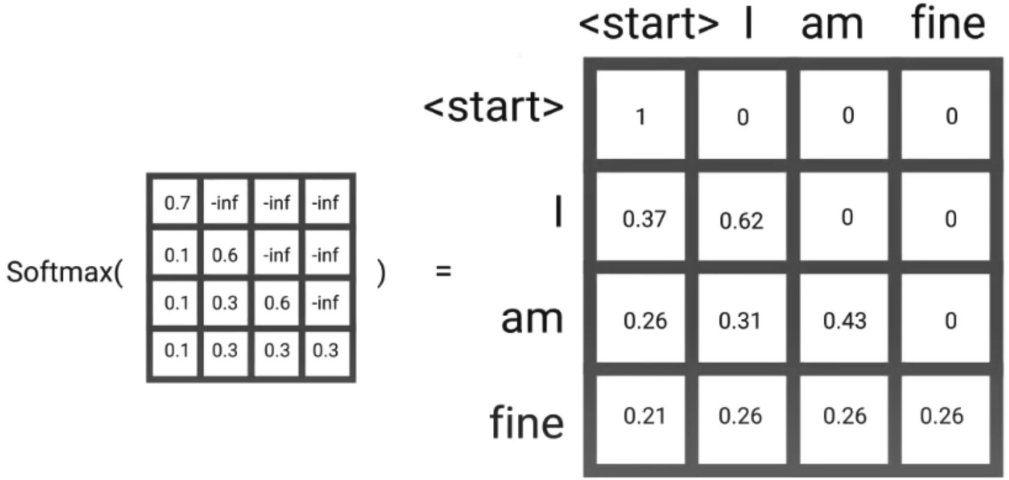
Attention in decoder very similar to attention in encoder, but:

- When processing the output sequence, we must only look into the past

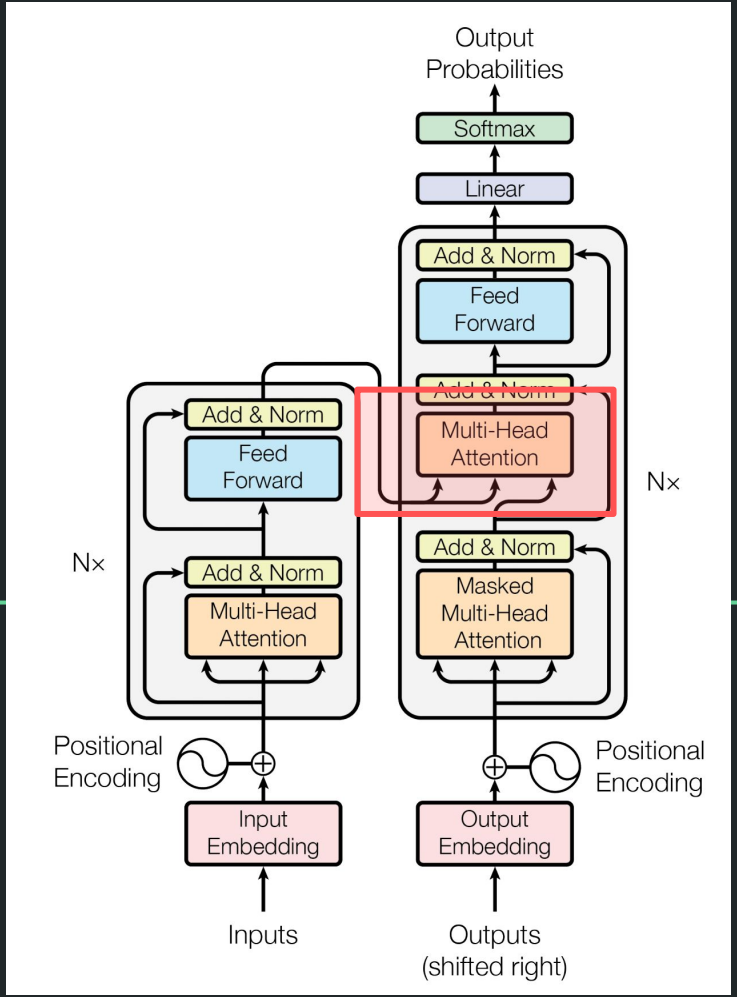
Masked (Multi-headed) Attention

Attention in decoder very similar to attention in encoder, but:

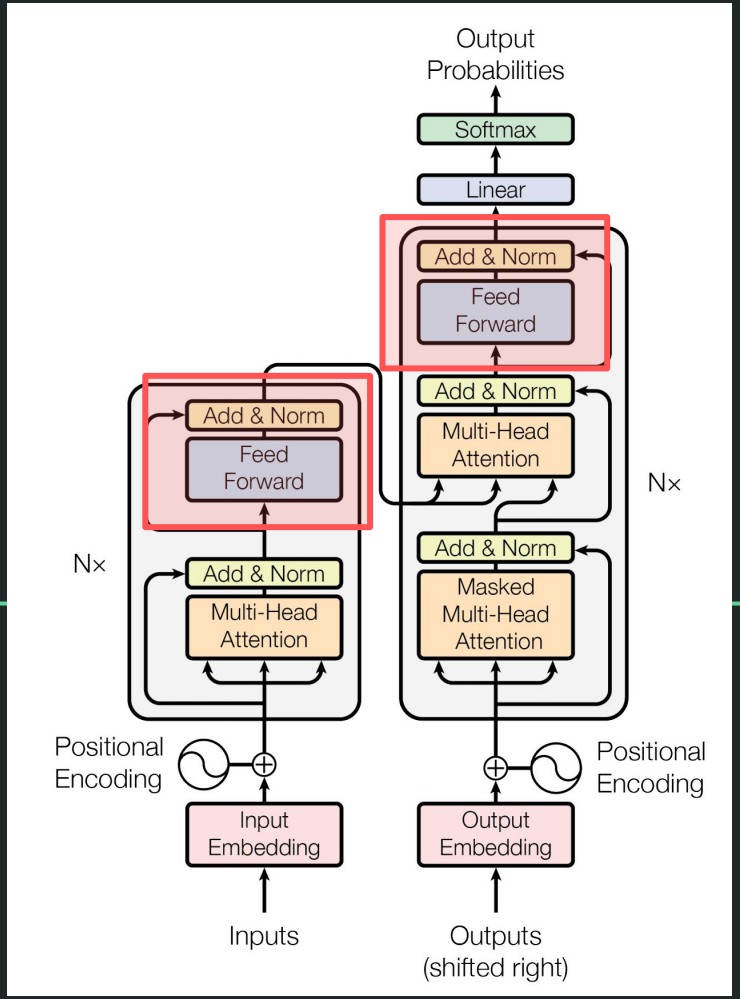
- When processing the output sequence, we must only look into the past



Attention between Input and Output Seq.

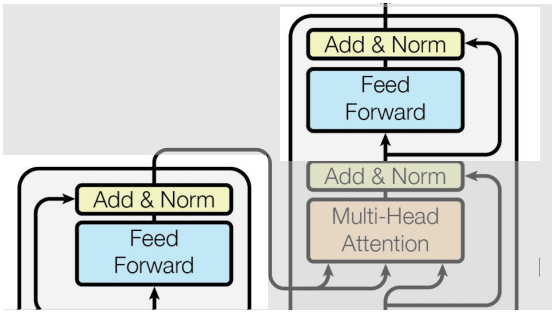


Feed Forward & Norm

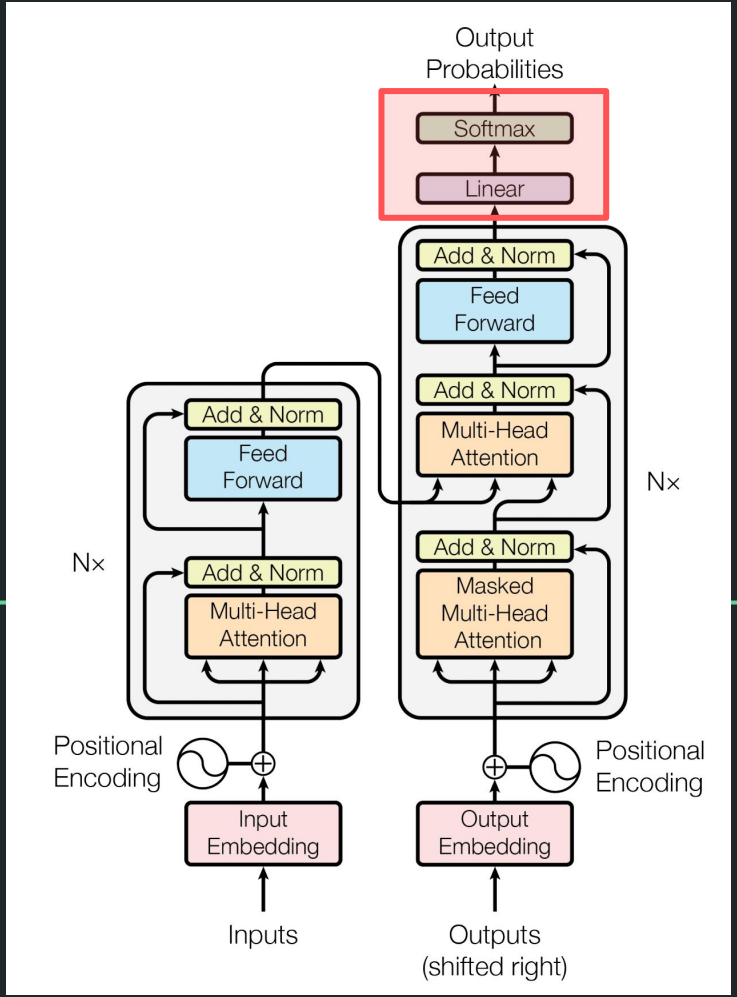


Feed Forward Network

- Feed forward Neural Network: used to transform the attention vectors into a format that is digestible by the next block
  - Allows parallelization: all input words can be passed in at the same time because attention blocks are independent of each other

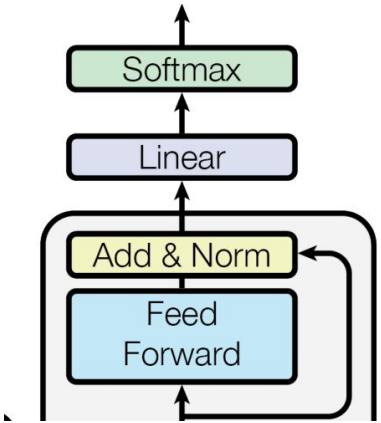


Linear & Softmax

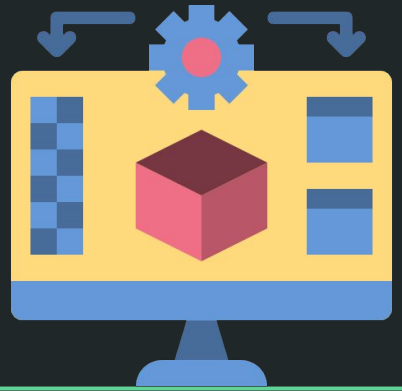


Output

- Linear Layer: Expand dimensions to vocabulary size in the output
- Softmax: Compute human-interpretable probabilities



Architectures



Encoder-decoder / Encoder-only / Decoder-only

Encoder-decoder:

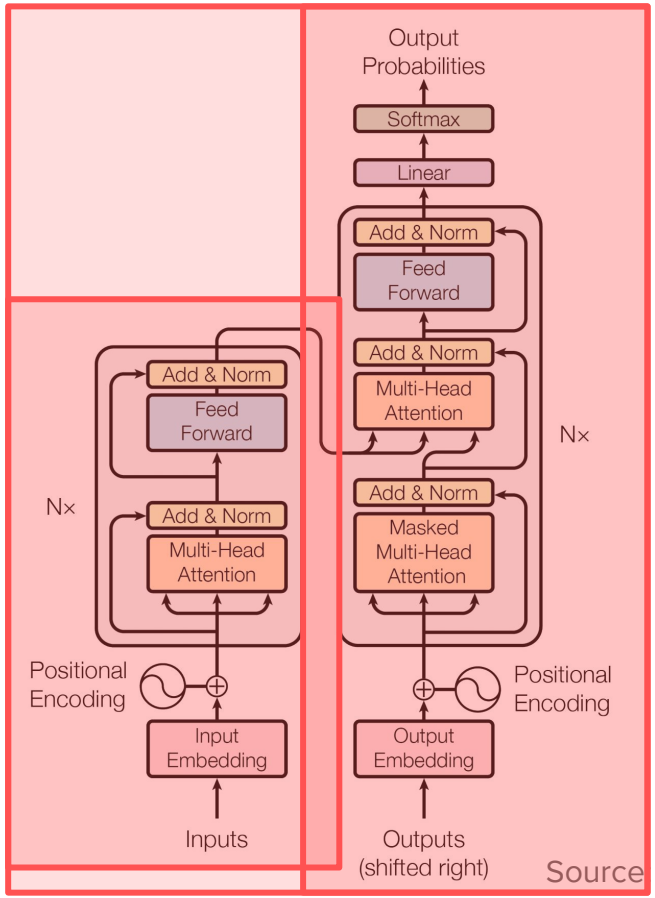
- Bidirectional encoder without masking
- Autoregressive decoder with masked attention

Encoder-only:

- Bidirectional encoder without masking
- Used to generate a dense embedding through unsupervised learning
- Typically is used with a fine-tuned head for downstream tasks, e.g., classification

Decoder-only:

- Just tokenization and positional embedding
- Autoregressive decoder with masked attention
- No cross-attention

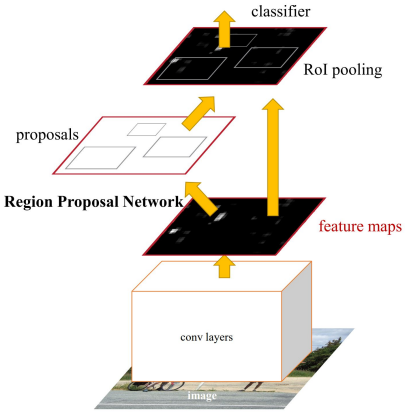


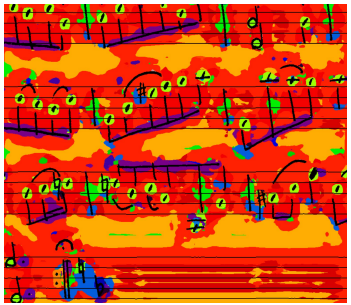
Source [2]

Transformers for Object Detection

State-of-the-art Object Detection

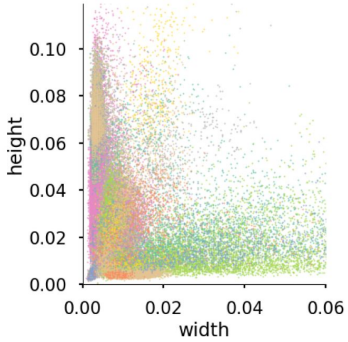
- Uses Deep Convolutional Neural Networks
- Problem can be formulated as surrogate regression and classification task:
  - Finding interesting regions in the image (“proposals”)
  - Classifying and refining these proposals
  - Formulated as
    - one-shot detector with a dense set of proposals (YOLO)
    - two-step process with sparse set of proposals (Faster R-CNN)
- Problem can be formulated as segmentation and clustering task:
  - Extract pixel-wise information
  - Group pixels into connected regions that depict to the same object

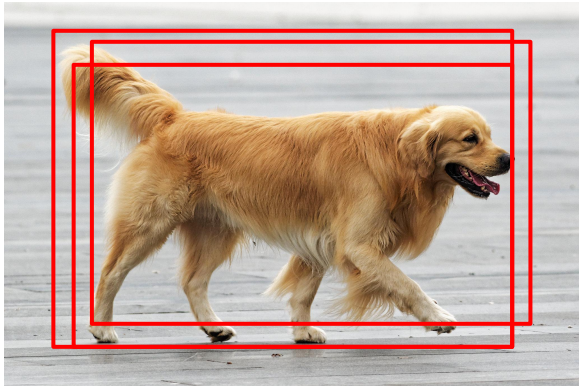




Problematic Issues

- Hand-designed method for anchor generation
  - Requires prior knowledge about the dataset
  - Heuristic to assign target box to anchors
- Non-maximum suppression for near-duplicate predictions
- How to handle fragmented objects in clustering-based approaches?



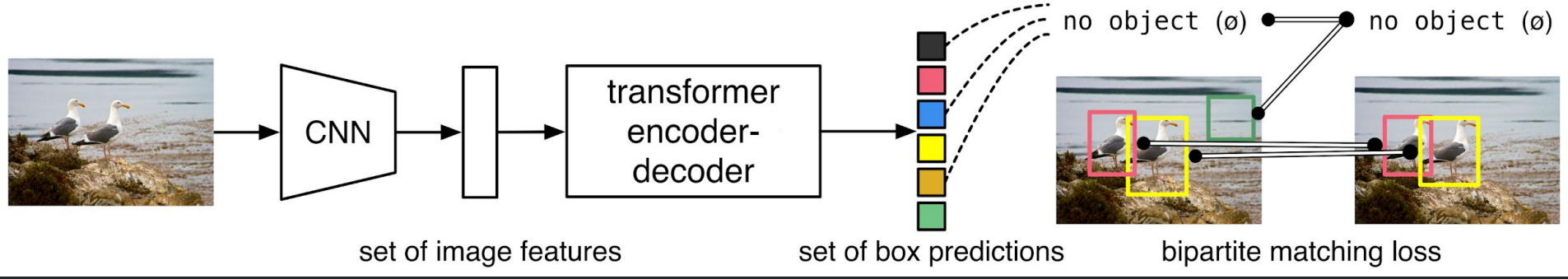


Ideally

Directly predict set of bounding boxes

DEtection TRansformer

End-to-End Object Detection with Transformers



Conventional CNN backbone (e.g., ResNet-50)

Source: [3]

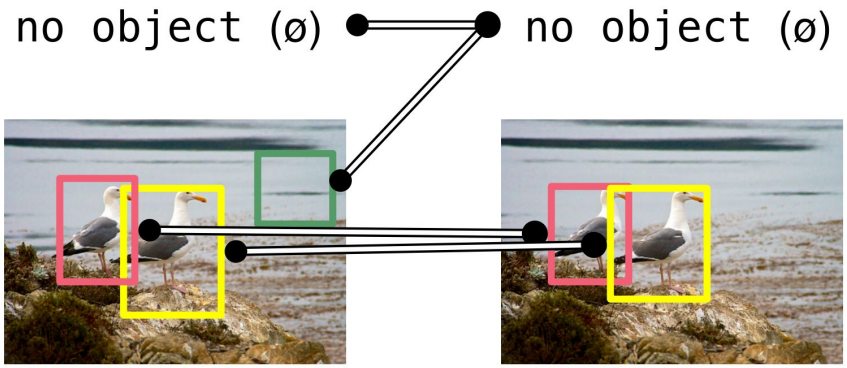
Bipartite Matching Loss

Challenge:

- Determine a unique matching between proposals and ground truth bounding boxes

Solution:

- Search for permutation with the lowest *pair-wise matching cost*
  - Taking into account class prediction and similarity of bounding boxes
- Efficiently computed using Hungarian algorithm

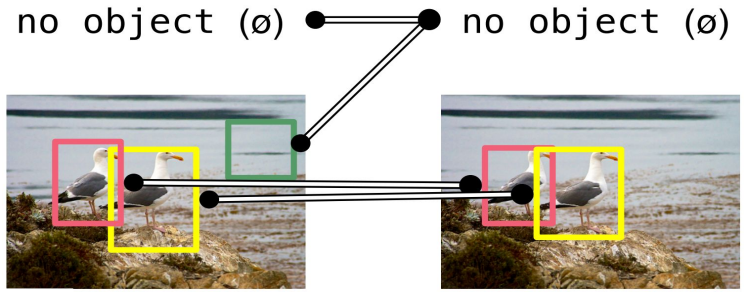


bipartite matching loss

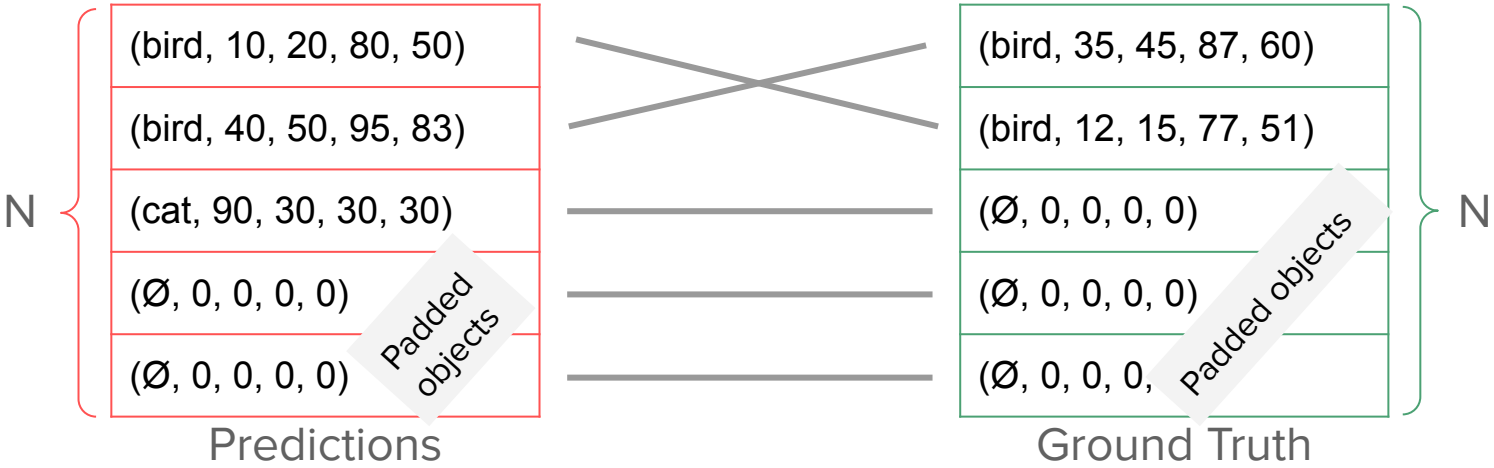
$$\hat{\sigma} = \arg\min_{\sigma \in \mathfrak{S}_N} \sum_{i}^{N} \mathcal{L}_{\text{match}}(y_i, \hat{y}_{\sigma(i)})$$

$$\mathcal{L}_{\text{match}}(y_i, \hat{y}_{\sigma(i)}) = -\mathbb{1}_{\{c_i \neq \varnothing\}} \hat{p}_{\sigma(i)}(c_i) + \mathbb{1}_{\{c_i \neq \varnothing\}} \mathcal{L}_{\text{box}}(b_i, \hat{b}_{\sigma(i)})$$

Bipartite Matching Loss - Example



bipartite matching loss



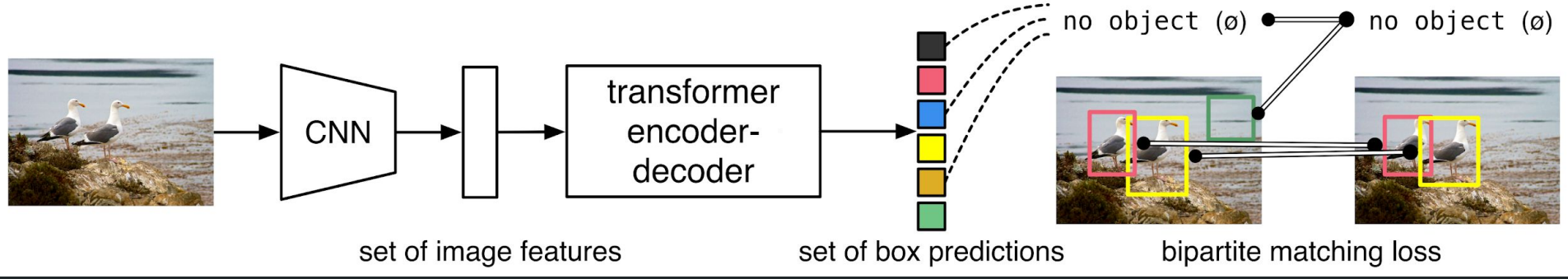
Bipartite Matching Loss

$$\mathcal{L}_{\text{Hungarian}}(y, \hat{y}) = \sum_{i=1}^{N} \Big[\underbrace{-\log \hat{p}_{\hat{\sigma}(i)}(c_i)}_{\text{Standard Cross-Entropy loss}} + \underbrace{\mathbb{1}_{\{c_i \neq \varnothing\}}}_{\text{No loss for 'No Object'-boxes}} \mathcal{L}_{\text{box}}(b_i, \hat{b}_{\hat{\sigma}}(i)) \Big]$$

$$\mathcal{L}_{\text{box}}(b_i, \hat{b}_{\sigma(i)}) = \lambda_{\text{iou}} \underbrace{\mathcal{L}_{\text{iou}}(b_i, \hat{b}_{\sigma(i)})}_{\text{Scale-invariant}} + \lambda_{\text{L1}} \underbrace{||b_i - \hat{b}_{\sigma(i)}||_1}_{\text{Scale-sensitive}}$$

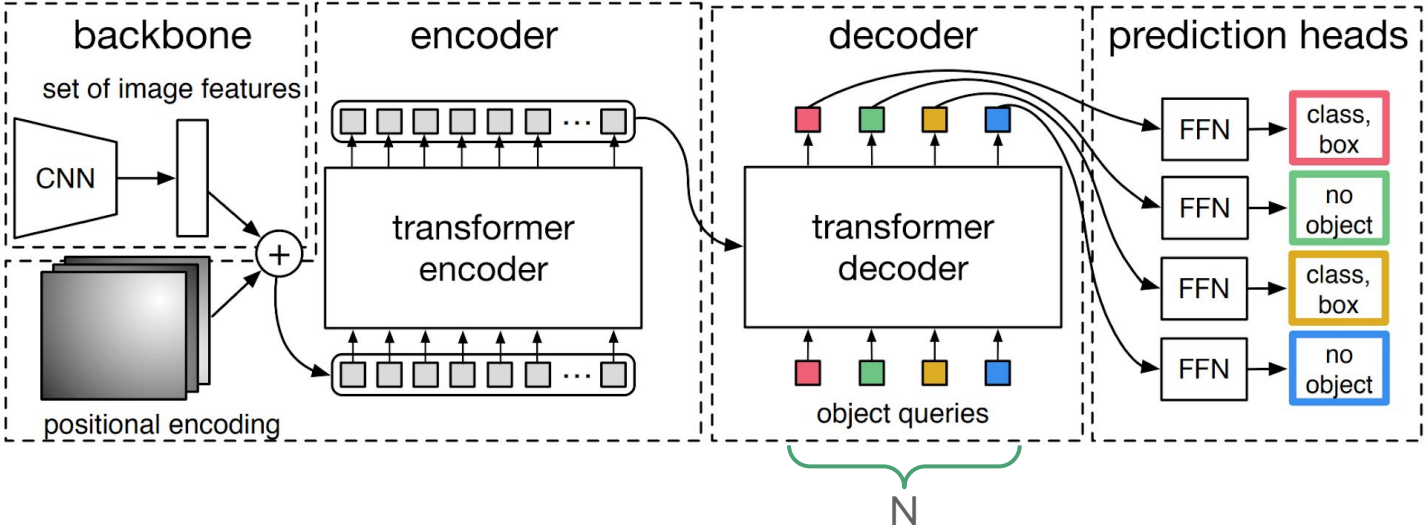
DEtection TRansformer

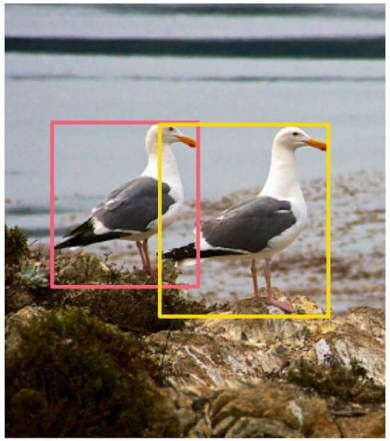
End-to-End Object Detection with Transformers



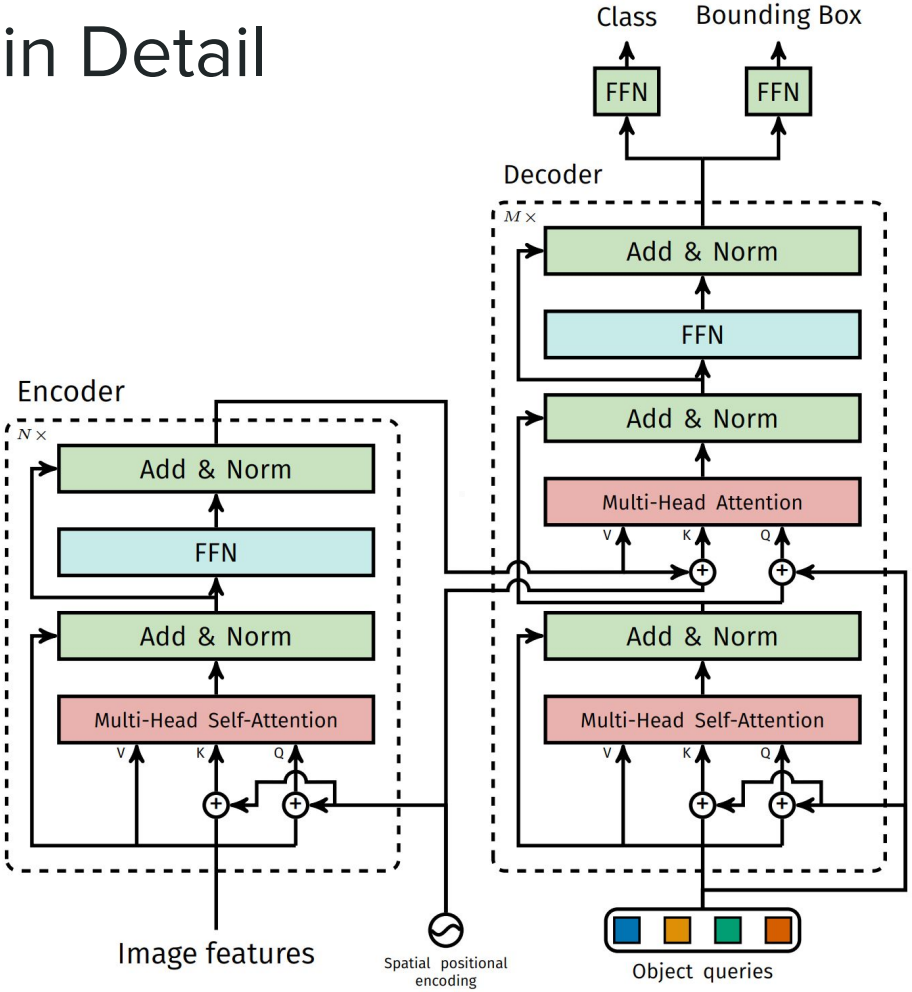
Transformer for Object Detection

- Similar to Transformers for NLP
- Instead of feeding back predictions and predicting one box at a time, all boxes are predicted simultaneously



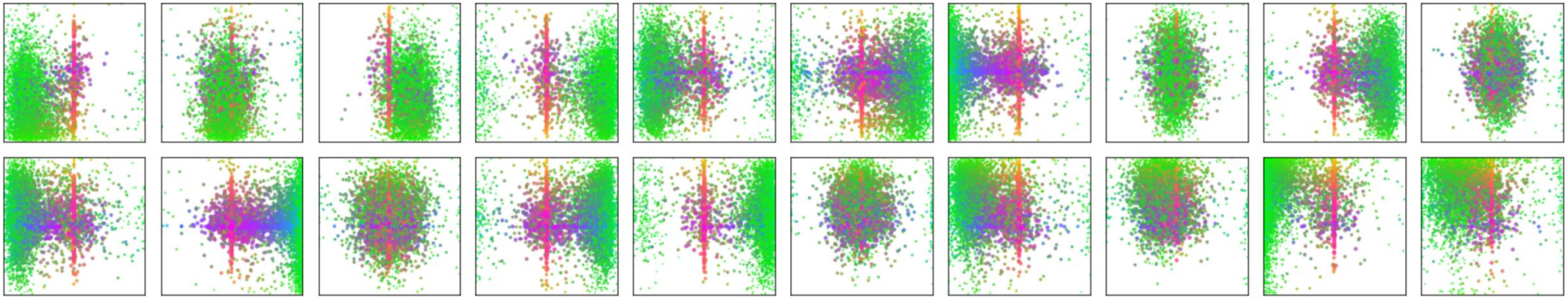


Architecture in Detail



Object Queries

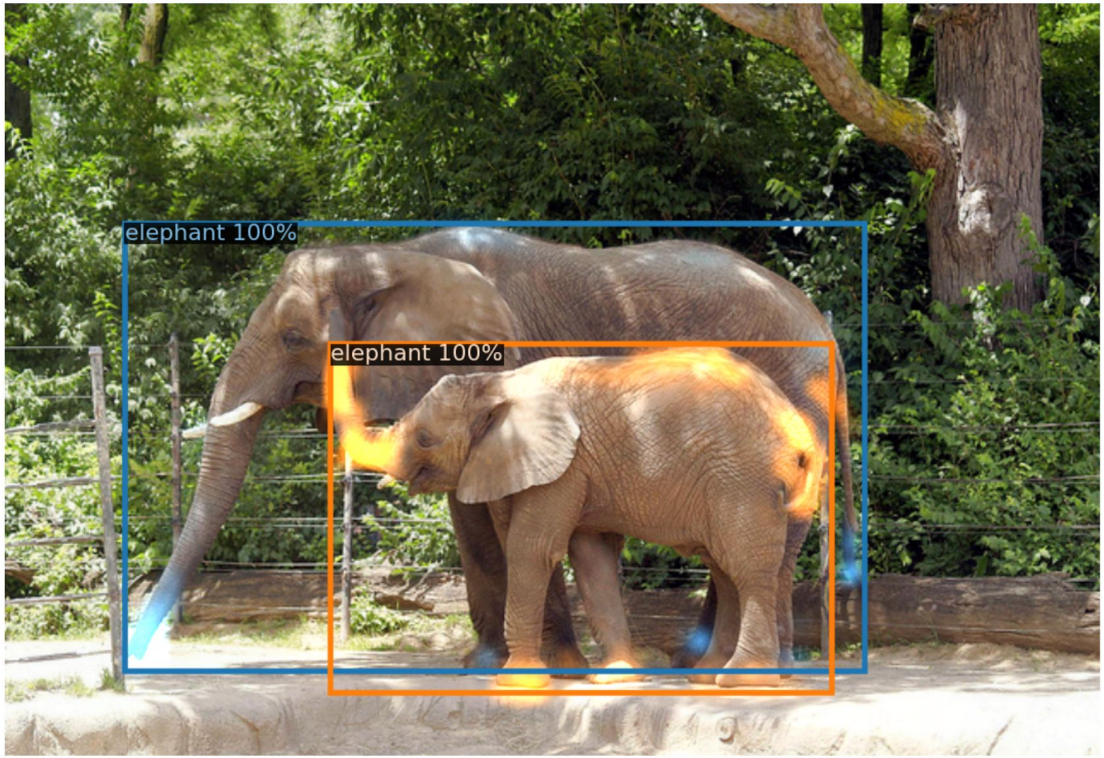
- Learnt object queries, that can “ask” different things
- Similar to guided anchoring

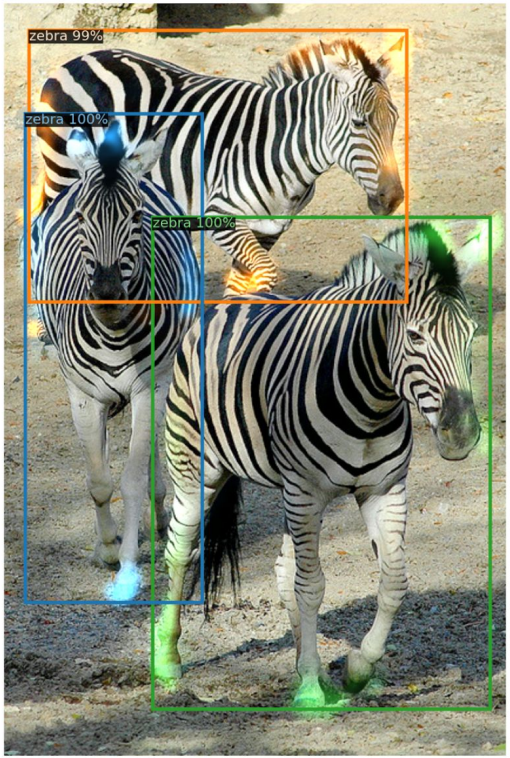


Visualization of all box predictions on all images from COCO 2017 val set for 20/100 prediction slots.

Green = Small boxes, Red = Large horizontal boxes, Blue = Large vertical boxes

Visualization of Decoder Attention



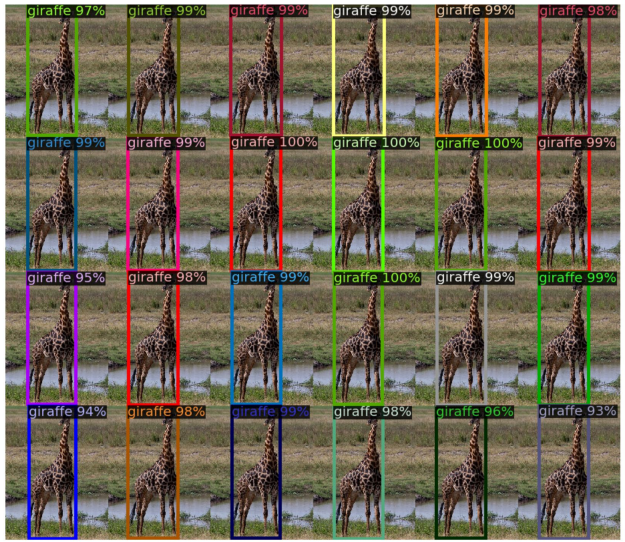


Inference code

```python
import torch
from torch import nn
from torchvision.models import resnet50

class DETR(nn.Module):

    def __init__(self, num_classes, hidden_dim, nheads,
                 num_encoder_layers, num_decoder_layers):
        super().__init__()
        # We take only convolutional layers from ResNet-50 model
        self.backbone = nn.Sequential(*list(resnet50(pretrained=True).children())[:-2])
        self.conv = nn.Conv2d(2048, hidden_dim, 1)
        self.transformer = nn.Transformer(hidden_dim, nheads,
                                          num_encoder_layers, num_decoder_layers)
        self.linear_class = nn.Linear(hidden_dim, num_classes + 1)
        self.linear_bbox = nn.Linear(hidden_dim, 4)
        self.query_pos = nn.Parameter(torch.rand(100, hidden_dim))
        self.row_embed = nn.Parameter(torch.rand(50, hidden_dim // 2))
        self.col_embed = nn.Parameter(torch.rand(50, hidden_dim // 2))

    def forward(self, inputs):
        x = self.backbone(inputs)
        h = self.conv(x)
        H, W = h.shape[-2:]
        pos = torch.cat([
            self.col_embed[:W].unsqueeze(0).repeat(H, 1, 1),
            self.row_embed[:H].unsqueeze(1).repeat(1, W, 1),
        ], dim=-1).flatten(0, 1).unsqueeze(1)
        h = self.transformer(pos + h.flatten(2).permute(2, 0, 1),
                             self.query_pos.unsqueeze(1))
        return self.linear_class(h), self.linear_bbox(h).sigmoid()

detr = DETR(num_classes=91, hidden_dim=256, nheads=8, num_encoder_layers=6, num_decoder_layers=6)
detr.eval()
inputs = torch.randn(1, 3, 800, 1200)
logits, bboxes = detr(inputs)
```

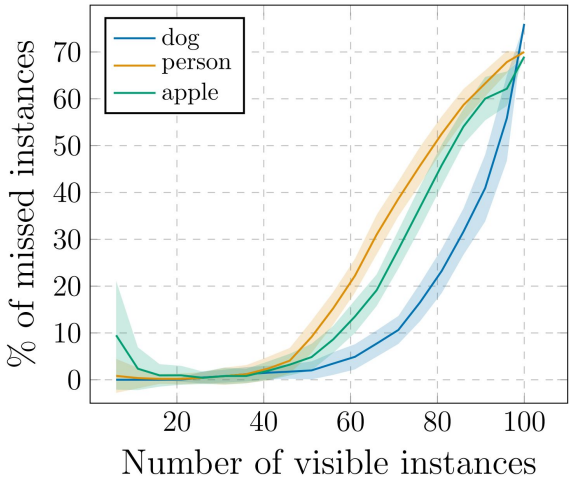
Strengths

- Generalizes well to unseen number of instances
- Can be extended to perform Panoptic Segmentation



Caveats

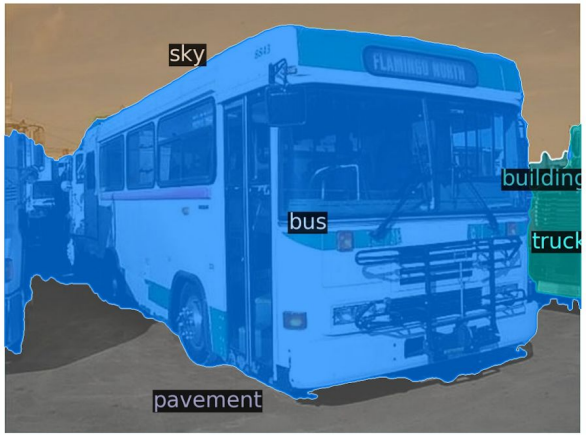
- Very long training schedule
- Does not perform very well on small objects
- Does not perform very well with many objects (coming near max. nr of obj.)
- But in general performs comparable or better than Faster R-CNN

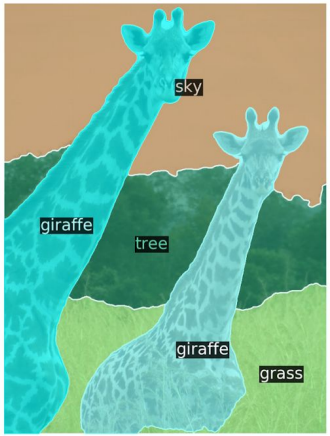


Model	GFLOPS/FPS	#params	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Faster RCNN-DC5	320/16	166M	39.0	60.5	42.3	21.4	43.5	52.5
Faster RCNN-FPN	180/26	42M	40.2	61.0	43.8	24.2	43.5	52.0
Faster RCNN-R101-FPN	246/20	60M	42.0	62.5	45.9	25.2	45.6	54.6
Faster RCNN-DC5+	320/16	166M	41.1	61.4	44.3	22.9	45.9	55.0
Faster RCNN-FPN+	180/26	42M	42.0	62.1	45.5	26.6	45.4	53.4
Faster RCNN-R101-FPN+	246/20	60M	44.0	63.9	**47.8**	**27.2**	48.1	56.0
DETR	86/28	41M	42.0	62.4	44.2	20.5	45.8	61.1
DETR-DC5	187/12	41M	43.3	63.1	45.9	22.5	47.3	61.1
DETR-R101	152/20	60M	43.5	63.8	46.4	21.9	48.0	61.8
DETR-DC5-R101	253/10	60M	**44.9**	**64.7**	47.7	23.7	**49.5**	**62.3**

Playing with DETR

Source code: https://github.com/facebookresearch/detr





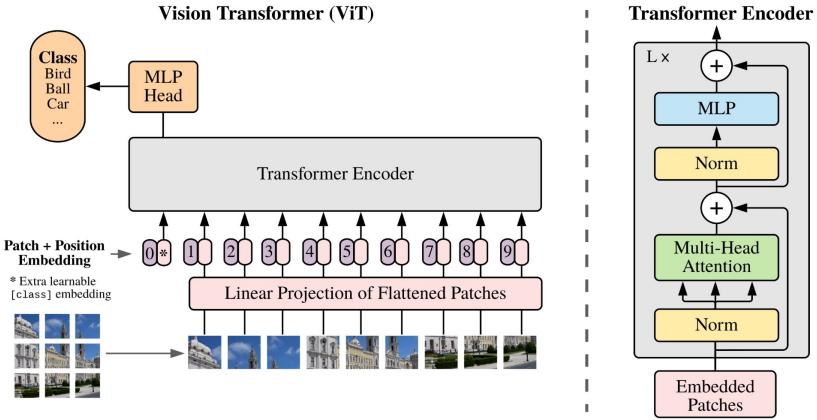
Advances in Transformers

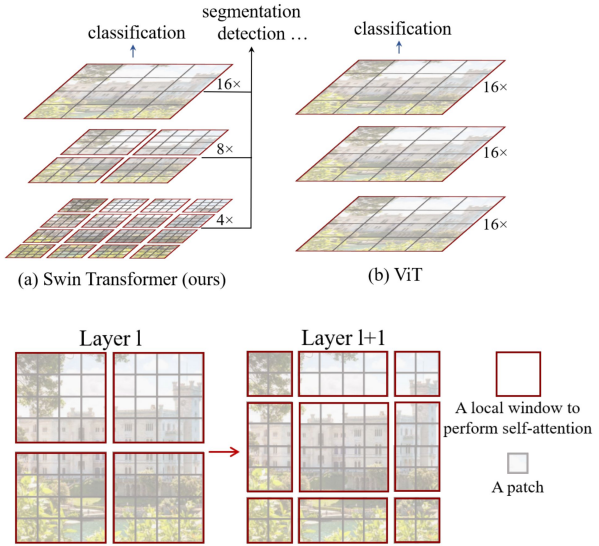
Advances in Transformers

- Embeddings from Language Models (ELMo): Replaces fixed embedding with context-aware embeddings [10]
- ULM-FiT: Process for fine-tuning language models for various tasks [11]
- BERT: Efficient pre-training by masking inputs in bidirectional transformers [12]
- Reformer: Local-sensitive hashing for more efficient attention [13]
- Linformer: Self-attention with linear complexity by approximating stochastic self-attention matrix with low-rank matrix [14]
- Lightweight Convolutions: Replacing self-attention with convolutions [15]

Advances in Vision Transformer

- Replacing Convolutions entirely with Transformers [16]
- Shifting the windows in Vision Transformers (Swin Transformers) [17, 18]
- Improving certain aspects of Detection Transformer [19]
- Using attention visualizations for explainability [20]





Summary

- Transformers are powerful models for sequence-to-sequence problems
- Transformers consist of encoder and decoder block with attention mechanisms
- (Self-)Attention allows the network to learn where to pay attention to
- Many current advances
  - More efficient pre-training
  - More efficient self-attention
- Popular architectural variants are
  - encoder-only architecture for training a dense embedding
  - decoder-only architecture for text generation
- Transformers can also be used for Object Detection by formulating it as a direct box prediction problem, transforming a set of object queries into unique bounding boxes

Literature

1. Halthor, Transformer Neural Networks Explained, 2020
2. Vaswani et al., Attention Is All You Need, 2017
3. Carion et al. End-to-End Object Detection with Transformers, 2020
4. Kilcher, End-to-End Object Detection with Transformers (Paper explanation), 2020
5. Pacha et al., A Baseline for General Music Object Detection with Deep Learning, 2018
6. Parmar et al. Image Transformer, 2018
7. Kilcher, Attention Is All You Need (Explained), 2017
8. Phi, Illustrated Guide to Transformers: Step by Step Explanation, 2020
9. Olah et al. Attention and Augmented Recurrent Neural Networks, 2016
10. Peters et al. Deep contextualized word representations, 2018
11. Howard et al. Universal Language Model Fine-tuning for Text Classification, 2018
12. Devlin et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2018
13. Kitaev et al. Reformer: The Efficient Transformer, 2020
14. Wang et al. Linformer: Self-Attention with Linear Complexity, 2020
15. Wu et al. Pay less attention with Lightweight and Dynamic Convolutions, 2019
16. Dosovitskiy et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale, 2021
17. Liu et al. Swin Transformer: Hierarchical Vision Transformer using Shifted Windows, 2021
18. Liu et al. Swin Transformer V2: Scaling Up Capacity and Resolution, 2022
19. Sun et al. Rethinking Transformer-based Set Prediction for Object Detection, 2021
20. Gildenblat, Exploring Explainability for Vision Transformers, 2021

Icon and Image credits

Free icons from Flaticon:

- https://www.flaticon.com/free-icon/asking_900415
- https://www.flaticon.com/free-icon/funnel_989330
- https://www.flaticon.com/free-icon/book_2490421
- https://www.flaticon.com/free-icon/medal_3176367
- https://www.flaticon.com/free-icon/microscope_3523823
- https://www.flaticon.com/free-icon/3d_5270902

Other images:

- https://www.vulture.com/2017/06/transformers-mythology-explained.html
- https://flic.kr/p/F3uXFa