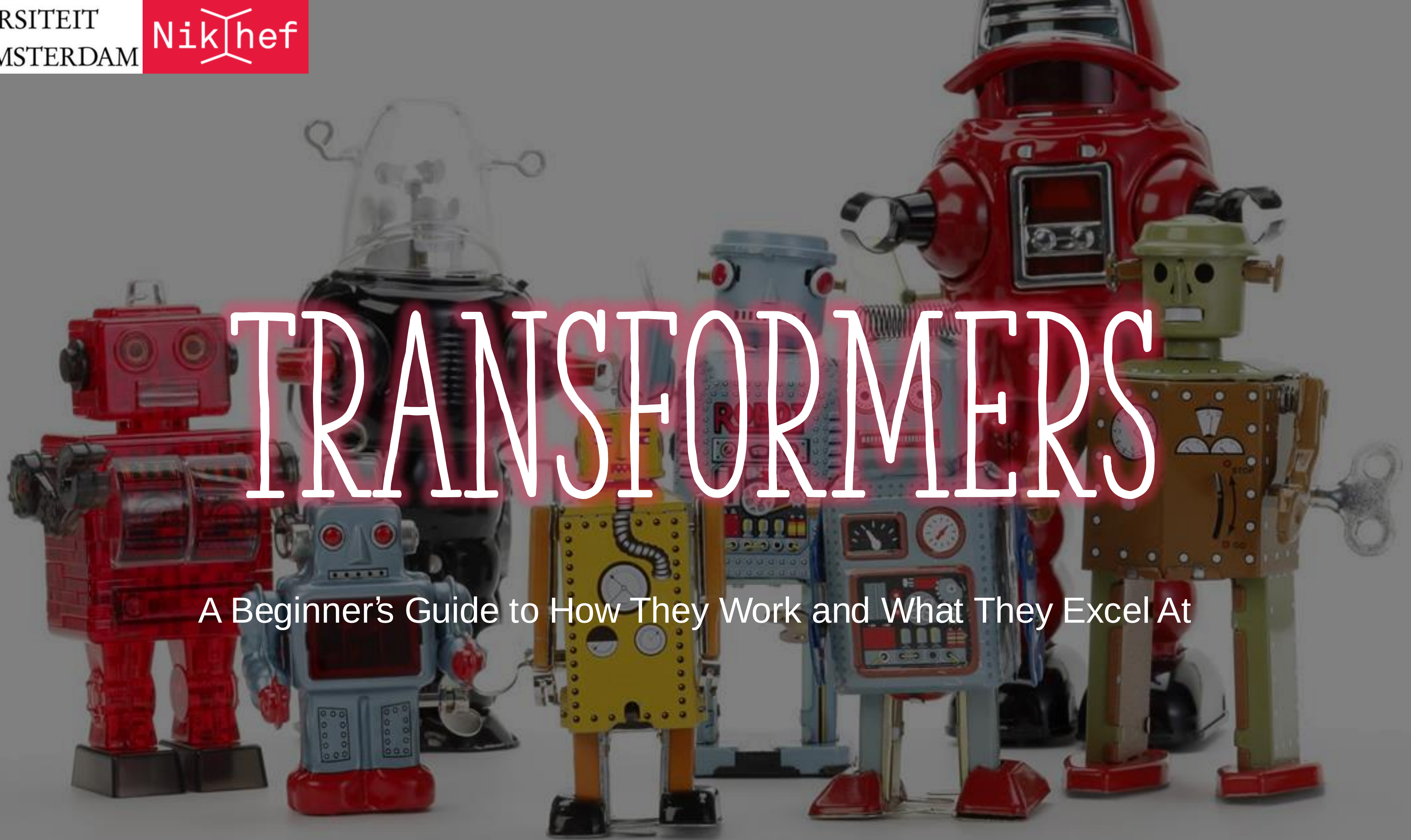
Universiteit van Amsterdam

Nikhef



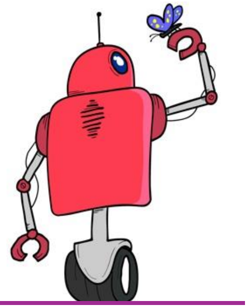
TRANSFORMERS

A Beginner's Guide to How They Work and What They Excel At

Workshop by Ambre Visive, Karel de Vries and Liza Cherepanova

January 31, 2025

Overview of the workshop



What is a transformer?

How do they work?

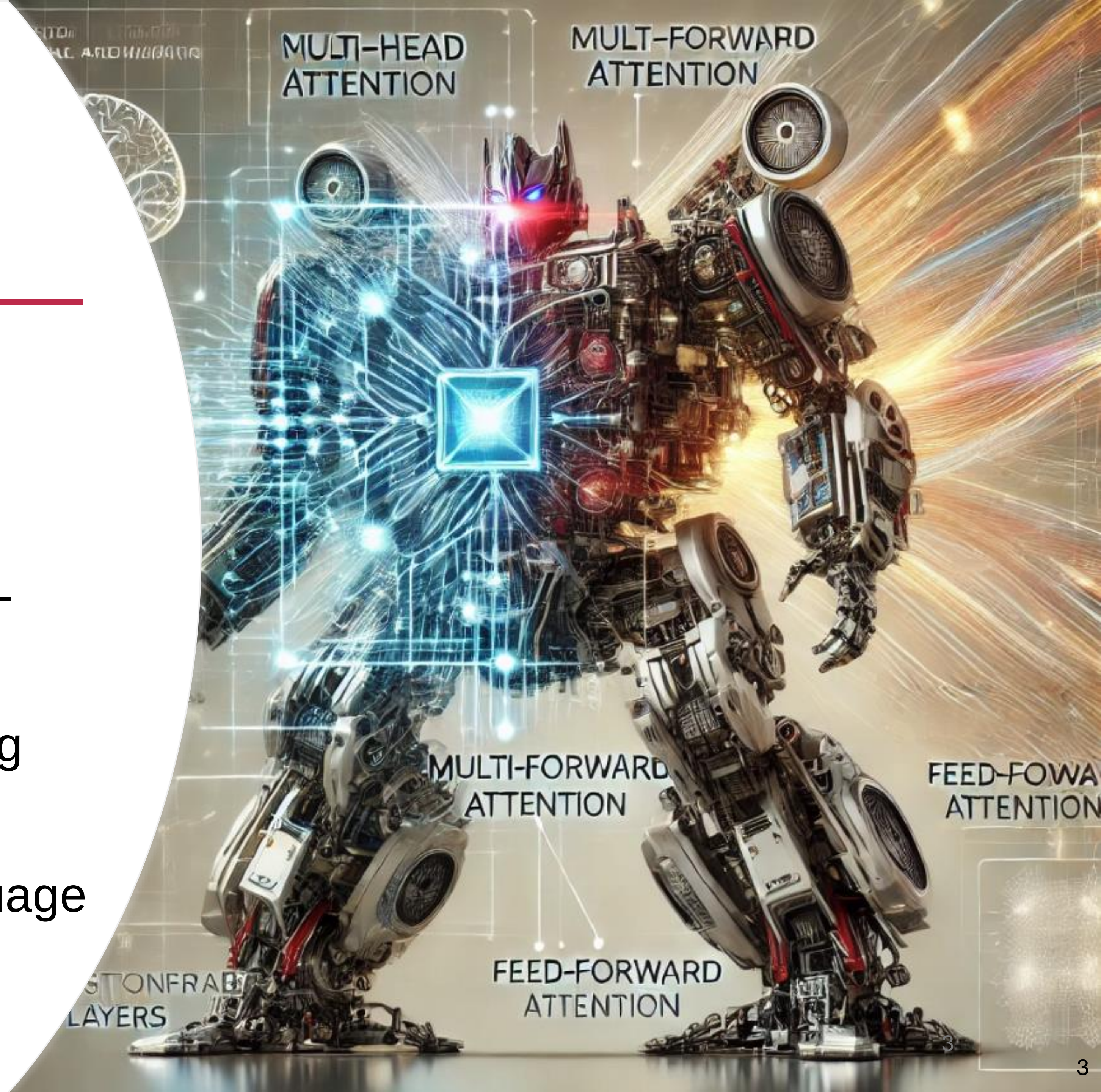
What are they good at?

Three talks from physicists who use them

Hands on tutorials on how to use them with **TensorFlow** and **PyTorch**

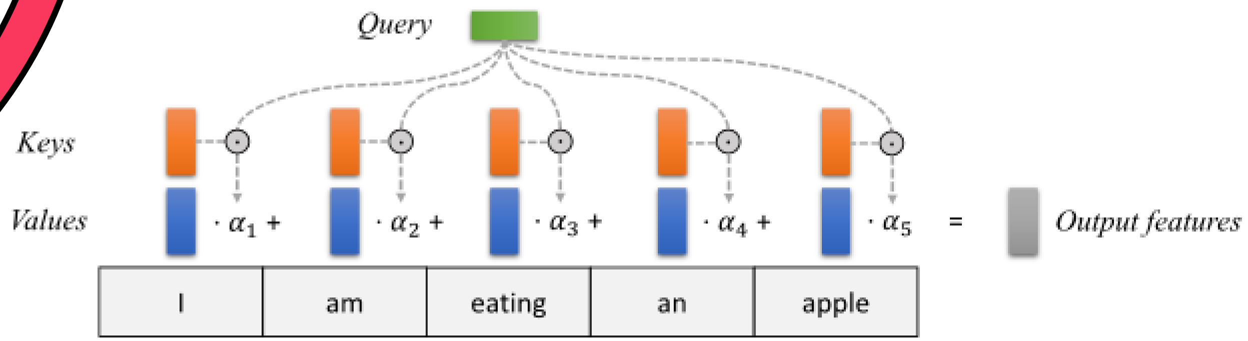
Transformers

- Introduced by Vaswani et al. in 2017 (“Attention Is All You Need”)[1].
- A deep learning model that uses self-attention to process sequential data.
- Enable parallel processing, improving scalability.
- Form the foundation for Large Language Models (LLM) like GPT and Copilot.



Key Concept of Transformer

Multi-Head-Attention

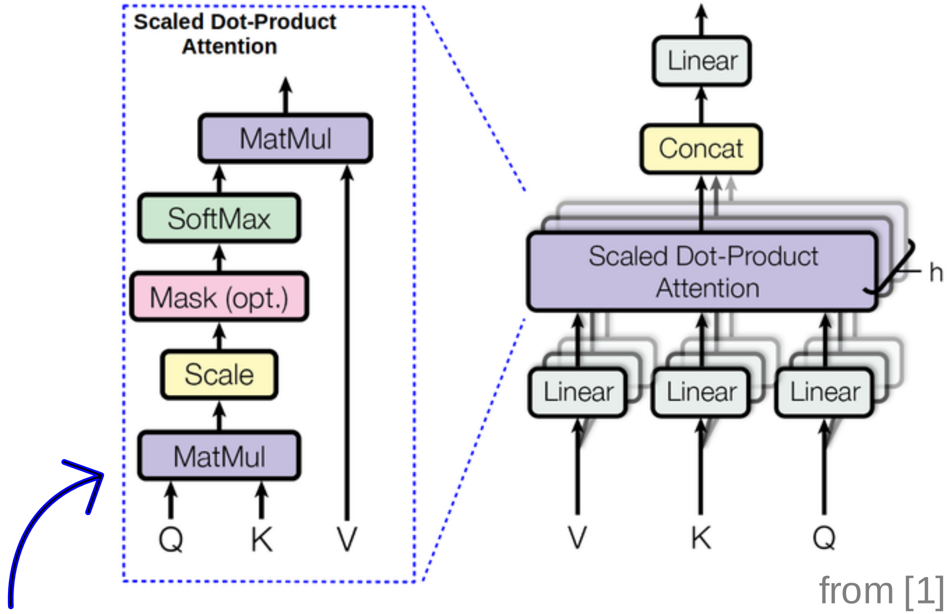


Attention: Query, Keys and Values

Self-attention: Every word has its own Query

Parallel heads

Capture context of sequential data



from [1]

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

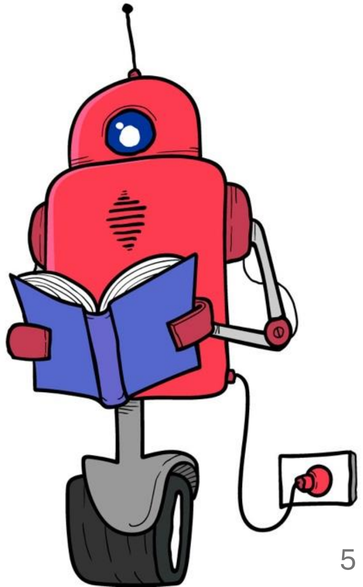
Concepts of Machine Learning you need to know to use transformers

Loss function

- Usually, compares the model output with a target
  gives a value for how bad the model performed
- *You choose the one most adapted to your use case*

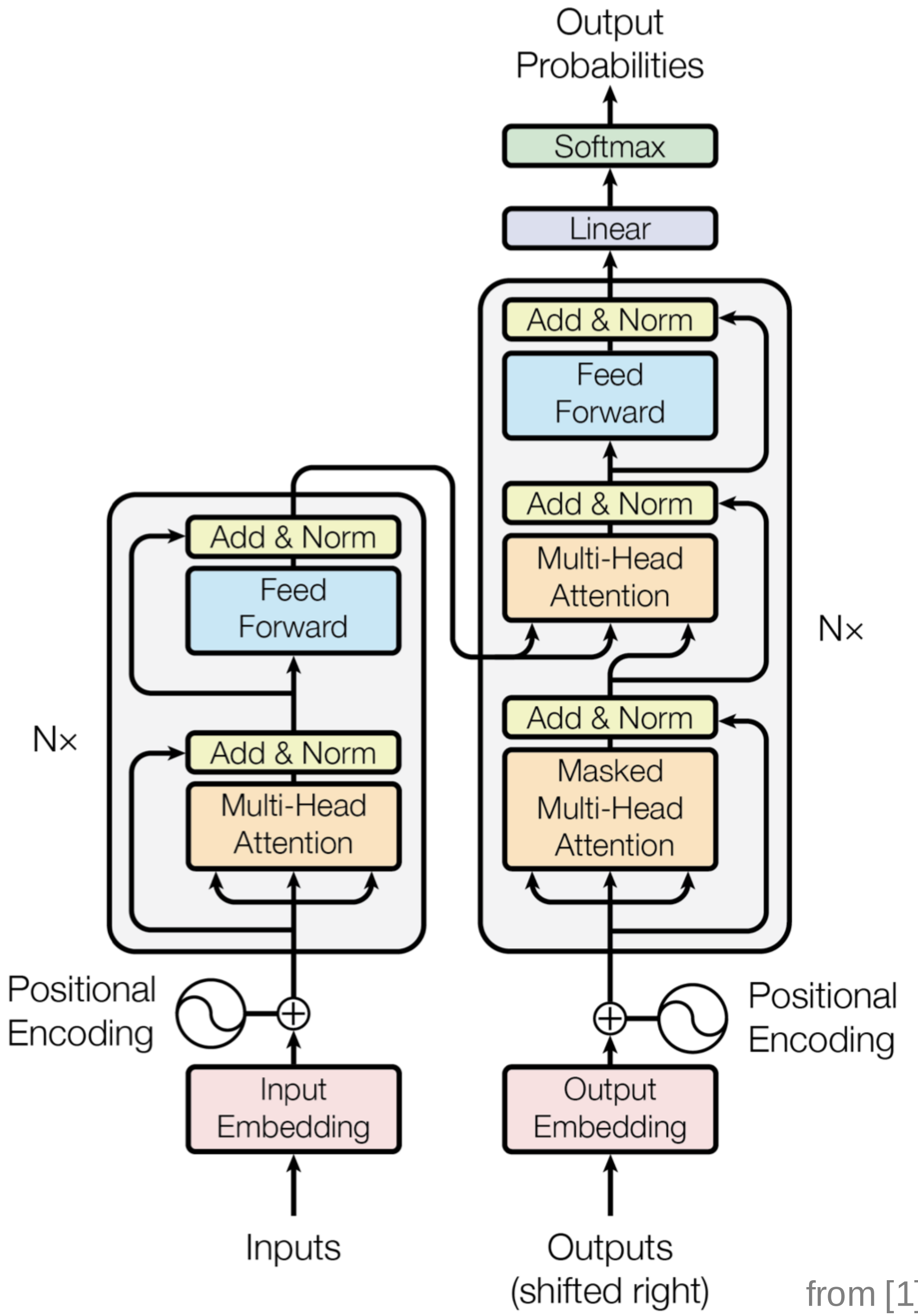
Optimizer

- try to find a minimal of the loss function
- updates the weights of the model



Architecture overview

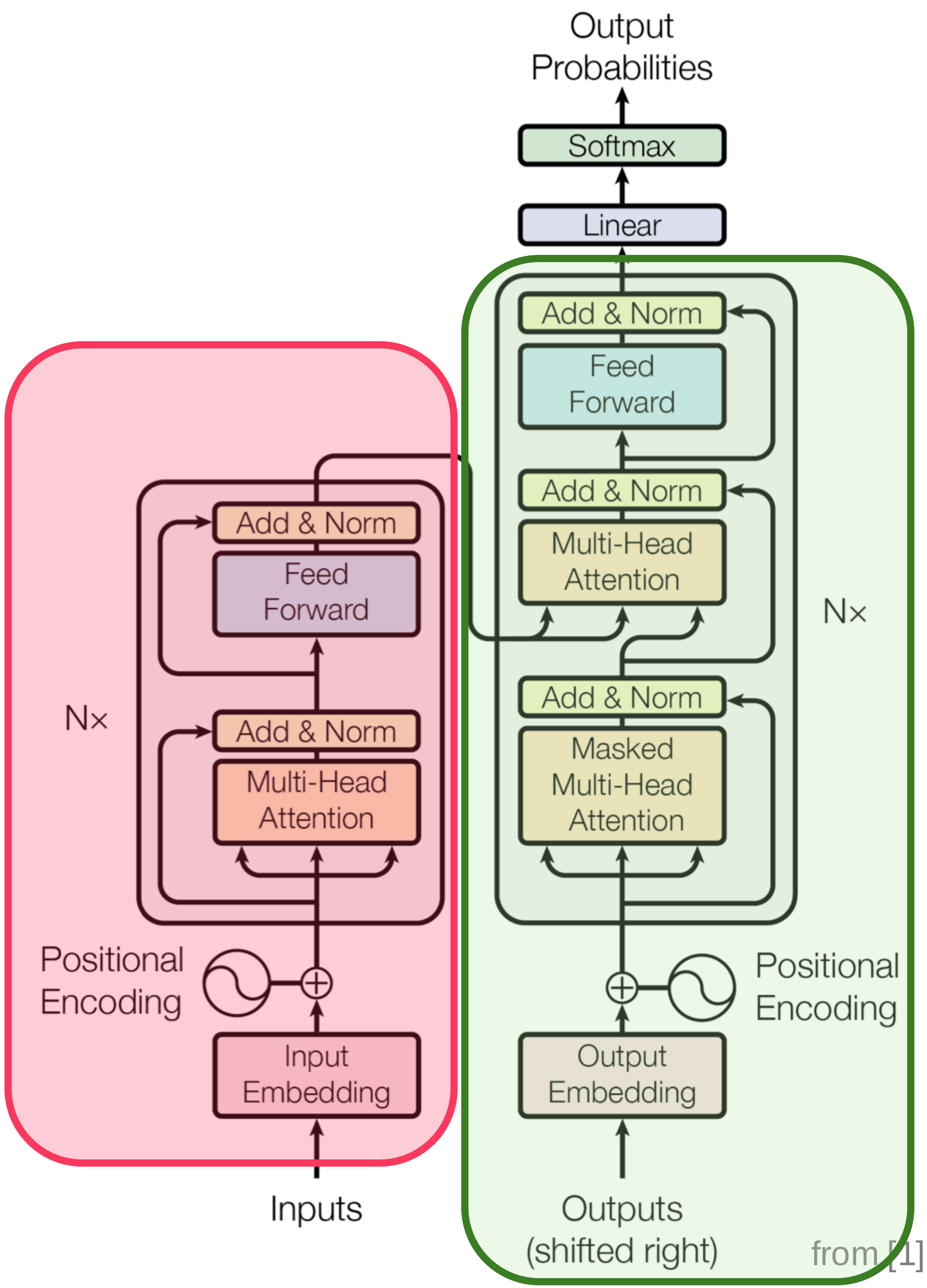
- Composed of several layers.
- Layers can be grouped into an Encoder and a Decoder part.
- Layers' sequence is repeated for every epoch



from [1]

Architecture overview

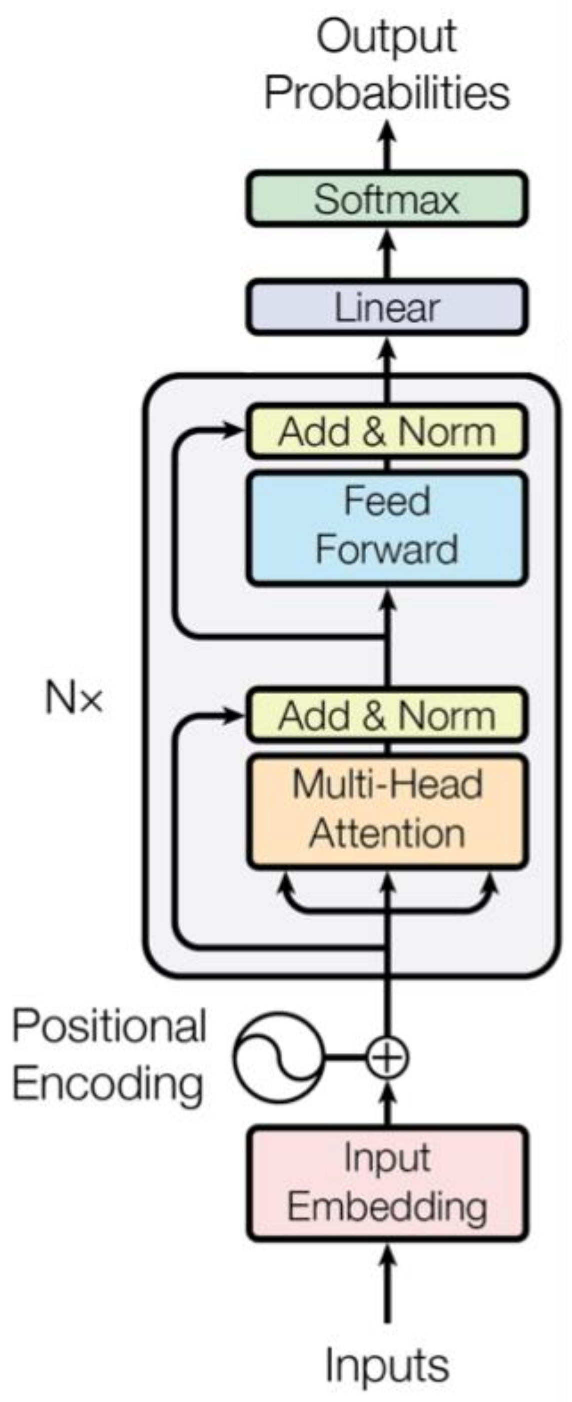
- Composed of several layers.
- Layers can be grouped into an Encoder and a Decoder part.
- Layers' sequence is repeated for every epoch



from [1]

Encoder-based Transformer

- **Input embedding**
  - converts the input tokens into dense vectors
- **Positional Encoding**
  - adds positional information about the order the token in the sequence to the dense vectors.
- **Multi-Head Attention = several Self-Attention Mechanisms in parallel**
  - Attention Mechanism allows the model to weigh the importance of different tokens in a sequence, capturing dependencies regardless of their distance
  - Parallelism capture the different aspects of the relationship between tokens
- **Feed-Forward Neural Network**
  - processes the output of the multi-head self-attention mechanism by looking at each token separately.
- **Normalization layers**
  - helps in stabilizing and improving the training process.
- **Linear Layer**
  - projects the output from the model to the desired number of output classes.
- **SoftMax Layer**
  - converts the raw scores from the linear layer into probabilities, which sum 100% over all raw scores.

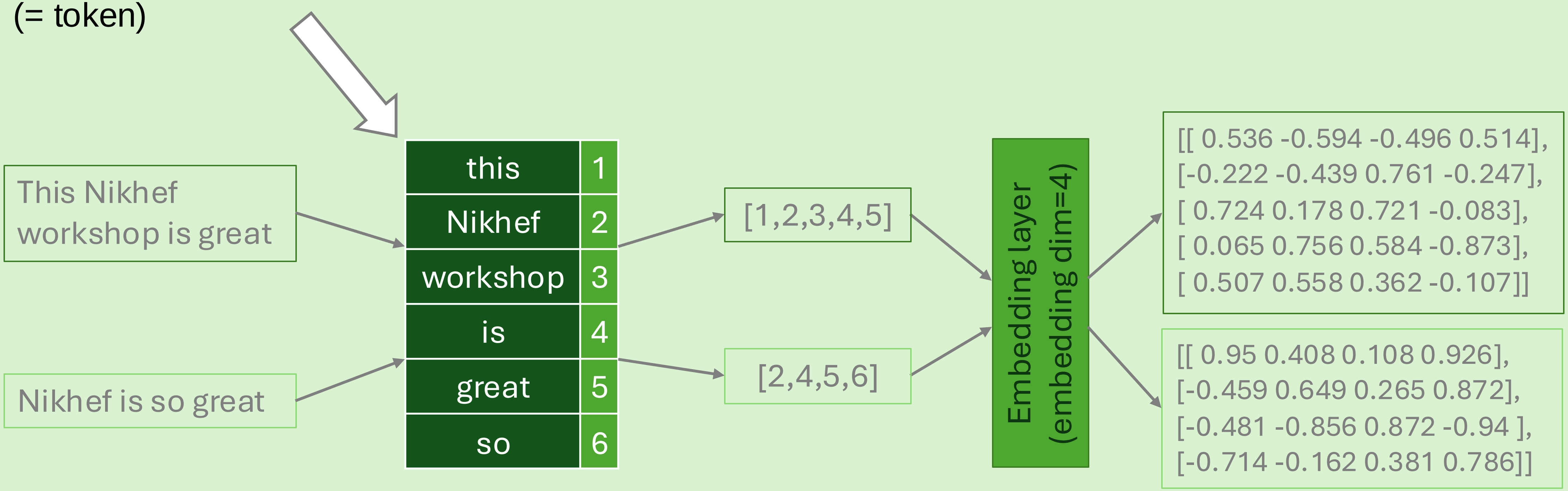


Tokenisation (insight to Large Language Model)

Step that comes in before the embedding of the data

Model knows that there is a limited number of possible tokens
(Think dictionary → limited number of words)

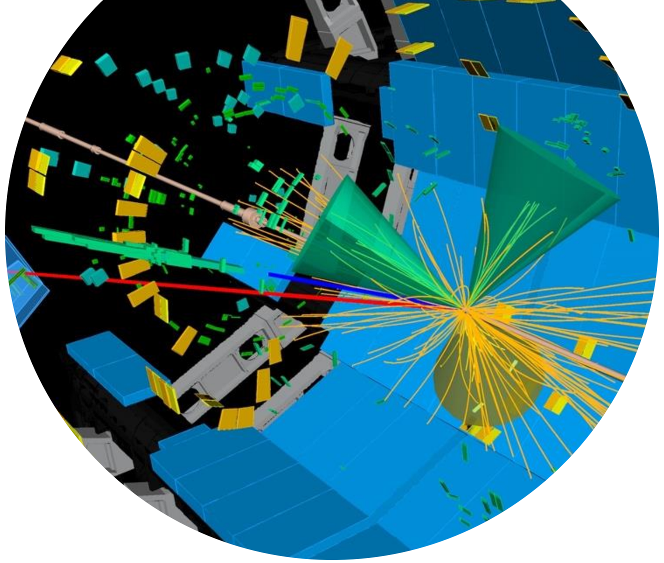
Whole idea = convert complicated data into a representation that the model can understand
(= token)

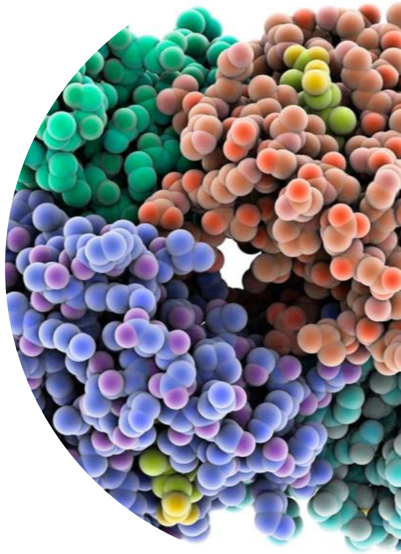


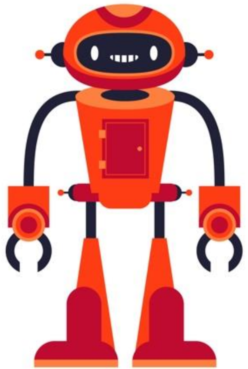
Thanks to this process, the model can understand and process complicated data which have meaning to us.

Why use transformers?

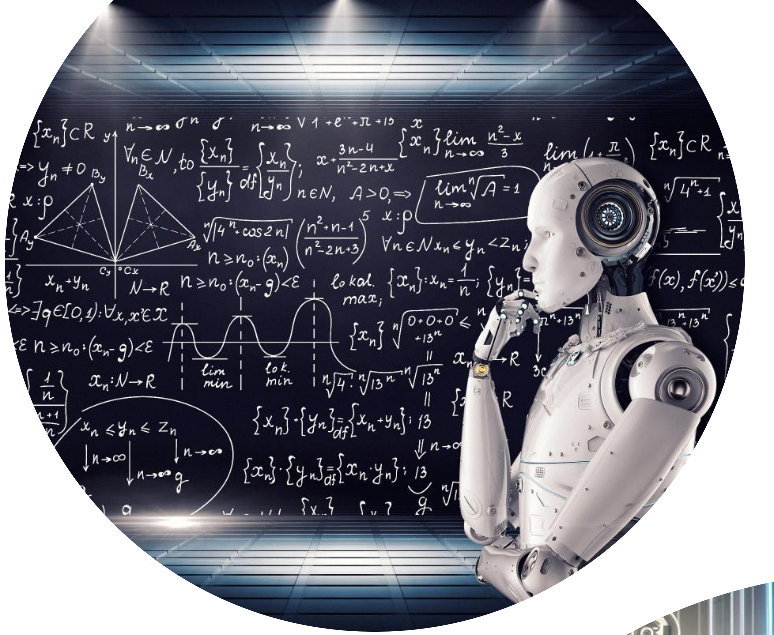
- Handle long-range dependencies in data effectively.
- Enable parallel data processing, speeding up training.
- Perform well with pre-trained models (e.g., transfer learning).
- Great for complicated data and tasks (e.g. particle collisions, drug discovery and text generation)







Downsides

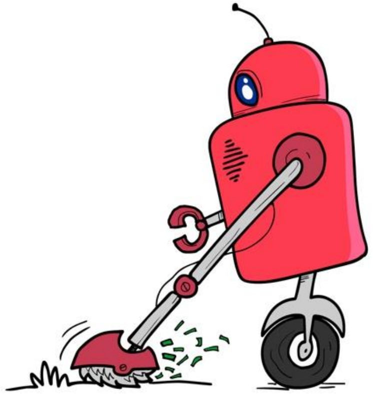


- Can be difficult to interpret (be careful with this)
- Can be costly to train
- Can require a big dataset to train on

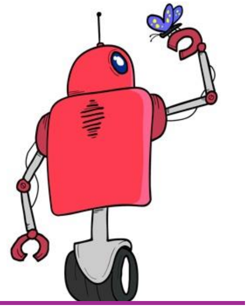
Conclusion

You know now the basics of transformers.

Before hearing how our colleagues are using it for and using transformers yourself, do you have any questions?



Overview of the workshop



What is a transformer?

How do they work?

What are they good at?

Three talks from physicists who use them

Hands on tutorials on how to use them with **TensorFlow** and **PyTorch**