Model Predictive Control and Reinforcement Learning
– Lecture 8: Transformers –

J. Hoffmann and Y. Zhang

University of Freiburg

October 9, 2023

Rise of Transformers

- Transformers were introduced in the paper "Attention is All You Need" [Vaswani et al. 2017]
- They are now used everywhere from Reinforcement Learning, Computer Vision to Natural Language Processing.

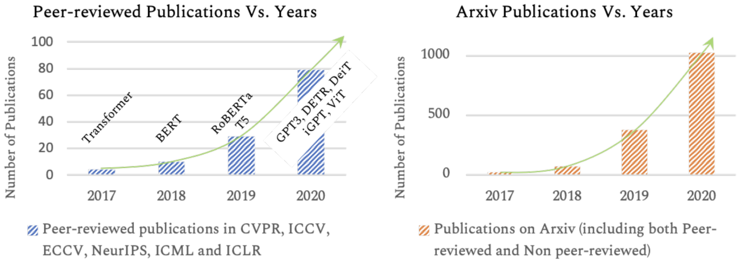
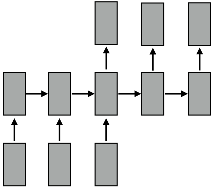


Figure: Drastic increase in the number of transformer-based papers.

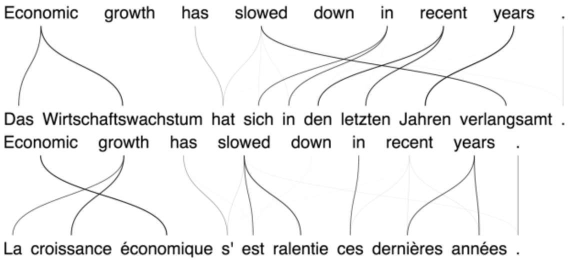
[Image source: Khan et al. 2021]

Motivating Example: Translation

many to many



e.g. Translation

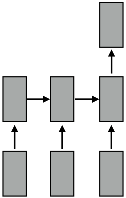


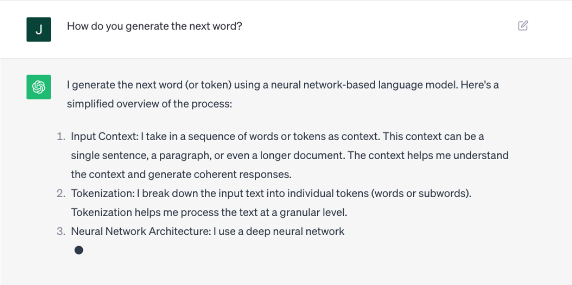
[Credit: nvidia]

Motivating Example: Text Generation (ChatGPT)

e.g. Text Generation

many to one





Recurrent Neural Networks

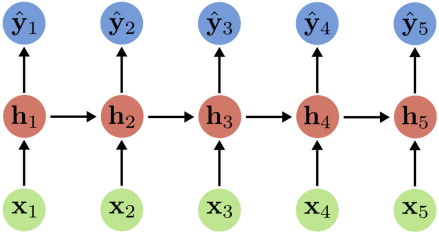


Figure: Recurrent Neural Network

Recurrent Neural Networks

$$h_t = f_h(h_{t-1}, x_t)$$
$$\hat{y}_t = f_y(h_t)$$

- Recurrent Neural Networks encode past information in their hidden state h.
- In theory, they can store information of arbitrary long sequences in h.
- However, they are are hard to train for long sequences (backprop through time).

[Image source: Geiger 2022]

Transformers are Autoregressive Models

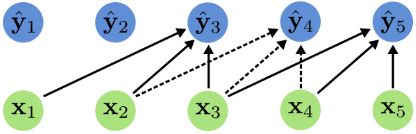


Figure: Autoregressive Model

Autoregressive Model

$$\hat{y}_t = f(x_t, x_{t-1}, \ldots, x_{t-N})$$

- An autoregressive model with block size N is a feedforward model which predicts the output $\hat{y}_t$ based on the last N previous variables $x_{t-1}, x_{t-2}, \ldots x_{t-N}$.
- Often a prediction $\hat{y}_t$ is the next steps input, $\hat{y}_t = x_{t+1}$, thus the term autoregressive.
- Assumption: Our prediction $\hat{y}_t$ is independent of $\{x_i \mid i < t - N\}$!

[Definition and image source: Geiger 2022]

Overview

1. Transformer
2. Attention Masks
3. Embedding
4. Applications

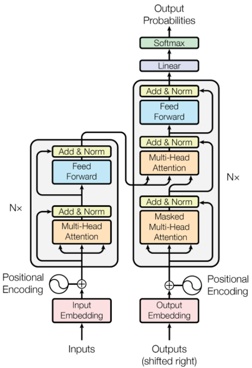


Figure: Original Transformer

[Image source: Vaswani et al. 2017]

Overview

1 Transformer

2 Attention Masks

3 Embedding

4 Applications

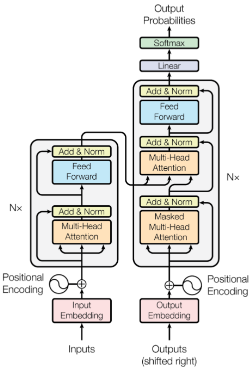


Figure: Original Transformer

[Image source: Vaswani et al. 2017]

High-Level View of the Encoder-Decoder Architecture

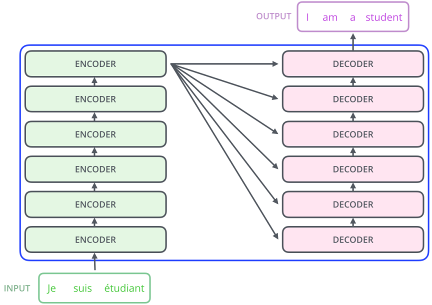


Figure: The encoder-decoder transformer architecture was designed for translation tasks.

- Encoder (decoder) blocks share the same architecture but have different trainable weights.

[Image source: `https://jalammar.github.io/illustrated-transformer/`]

Zooming into the Encoder and Decoder Blocks

- Encoder and decoder blocks share two main components:
  - Self-Attention Layer
  - Feed Forward (Fully Connected Layer)
- The third component allows the decoder to focus on relevant parts of the input sentence:
  - Encoder-Decoder Attention

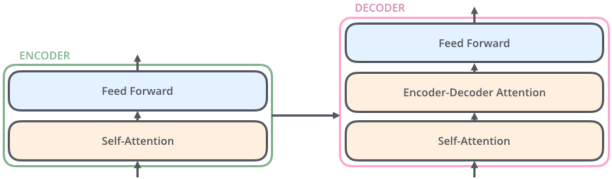


Figure: Components of the encoder and decoder blocks.

[Image source: `https://jalammar.github.io/illustrated-transformer/`]

Flow of Vectors Through the Encoder

- Each input is encoded into a vector $x_i \in \mathbb{R}^{1 \times d_\mathrm{e}}$ (e.g. $d_\mathrm{e} = 512$).
- An encoder block takes an input vector and outputs a vector with the same dimension d_e.

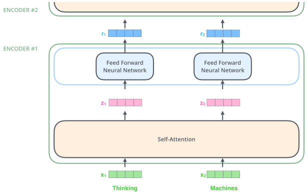


Figure: How vectors are processed in an encoder layer.

- The self-attention block operates on all inputs jointly.
- The feedforward block operates on each word separately.
- The vector of each word gets transformed to take into account the entire sentence.

[Adapted from: Foundations of Deep Learning (Hutter and Valada)]

Self-Attention: Attention as Soft Retrieval from a Database

- Assume we have a list of keys $K \in \mathbb{R}^{N \times d_\text{k}}$, a list of values $V \in \mathbb{R}^{N \times d_\text{v}}$ and a single query $q \in \mathbb{R}^{1 \times d_\text{k}}$.
- We denote with b_i, the i-th row vector of a matrix $B \in \mathbb{R}^{\text{d} \times \text{d}}$.
- Database retrieval: compare q to keys and try to find the exact match

$$\text{retrieval}(q, K, V) = \textstyle\sum_{i=1}^{N} \mathbb{1}_{q=k_i} \, v_i$$

- Attention: compare q to keys and return weighted average of values

$$\text{attention}(q, K, V) = \textstyle\sum_{i=1}^{N} a_i \, v_i,$$

where the weight is calculated by the softmax of the inner product:

$$a_i = \frac{\exp\left(q k_i^T / \sqrt{d_k}\right)}{\sum_{j=1}^{N} \exp\left(q k_j^T / \sqrt{d_k}\right)}.$$

k_1	v_1
k_2	v_2
.	.
.	.
.	.
k_N	v_N

[Adapted from: Foundations of Deep Learning (Hutter and Valada)]

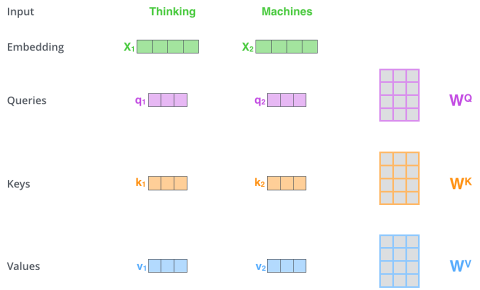


Figure: A self-attention block induces 3 trainable weight matrices (W^Q, W^K, W^V), that linearly transforms inputs x_i to yield q_i, k_i and v_i.

- Linear mapping for key and query:

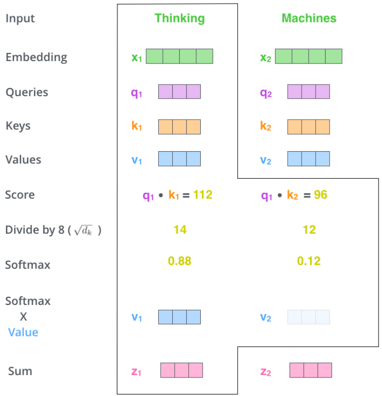
$$\begin{aligned} q_i &:= x_i W^Q \\ k_i &:= x_i W^K \end{aligned} \quad \text{with } W^Q, W^K \in \mathbb{R}^{d_\mathrm{e} \times d_\mathrm{k}}$$

- Linear mapping for value:

$$v_i := x_i W^V \quad \text{with } W^V \in \mathbb{R}^{d_\mathrm{e} \times d_\mathrm{v}}$$

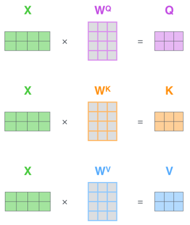
[Image source: `https://jalammar.github.io/illustrated-transformer/`]

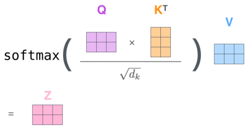
Self-Attention: Exemplary Calculation of Self-Attention



[Image source: https://jalammar.github.io/illustrated-transformer/]

Self-Attention: Matrix Form





1. Calculate query, key and value for $X \in \mathbb{R}^{N \times d_\mathrm{e}}$:

$$K = XW^K \in \mathbb{R}^{N \times d_\mathrm{k}}$$
$$Q = XW^Q \in \mathbb{R}^{N \times d_\mathrm{k}}$$
$$V = XW^V \in \mathbb{R}^{N \times d_\mathrm{v}}$$

2. Calculate softmax attention scores row-wise:

$$A = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) \in \mathbb{R}^{N \times N}$$

3. Apply "soft retrieval":

$$Z = AV \in \mathbb{R}^{N \times d_\mathrm{v}}$$

[Image source: `https://jalammar.github.io/illustrated-transformer/`]

Self-Attention: Attending to more than one concept?

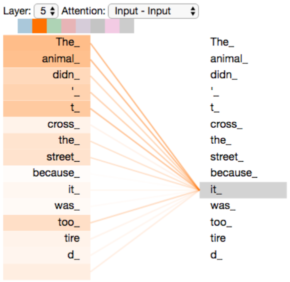


Figure: Single Attention

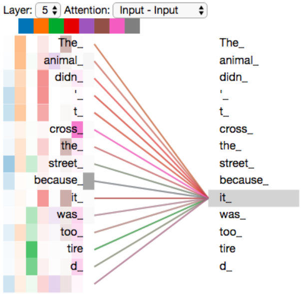
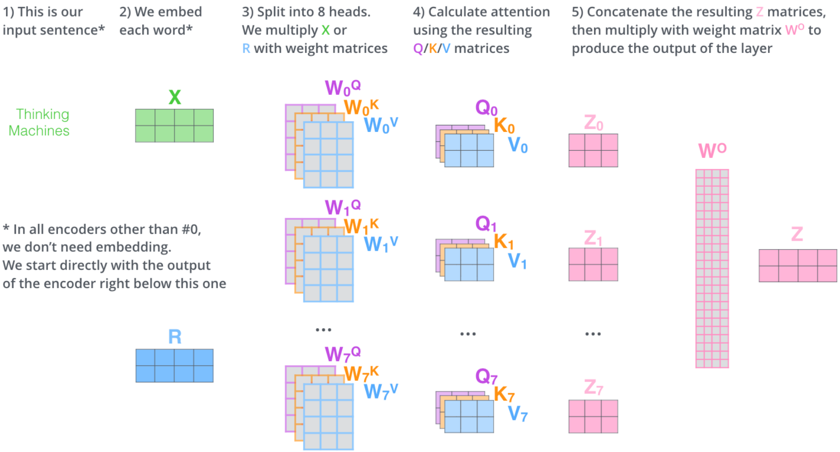


Figure: Multi-headed Attention

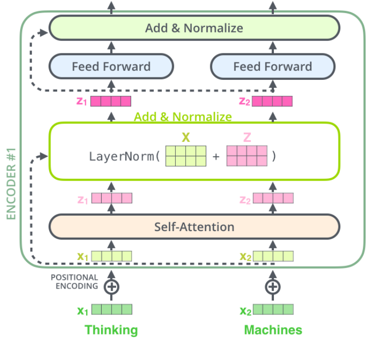
- More heads might lead to better training dynamics as indicated in Michel et al. 2019.

[Image source: `https://jalammar.github.io/illustrated-transformer/`]

Self-Attention: Multi-headed Attention



Finalizing the Encoder Block



Skip Connections (dashed lines):

- Are of the form $y = \text{layer}(x) + x$
- Allow information to bypass intermediate layers
- No vanishing gradients and "network can choose its own depth"

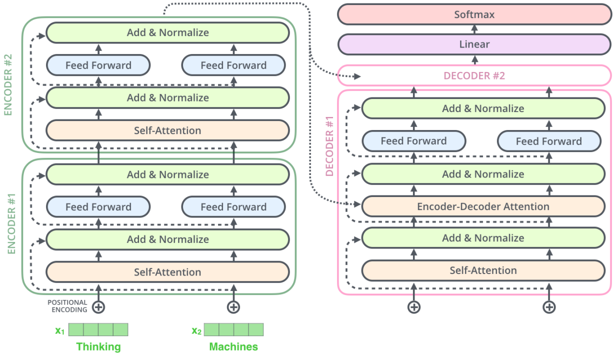
LayerNorm:

- Normalizes features based on all outputs of one layer
- Leads to more stable and faster training

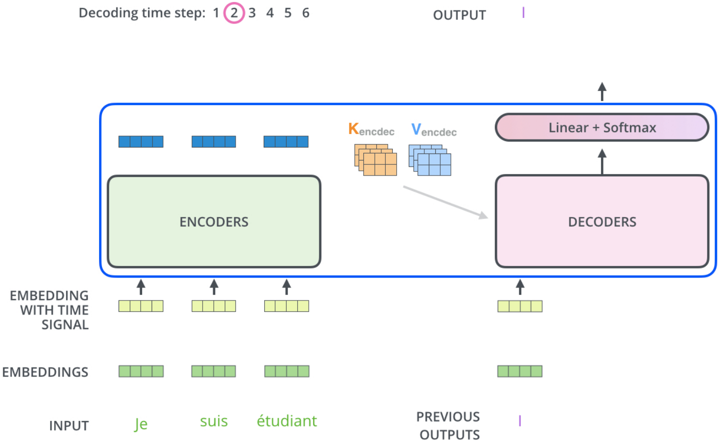
[Image source: `https://jalammar.github.io/illustrated-transformer/`]

Wrapping up the Encoder-Decoder Architecture

- Encoder and decoders are very similar.
- Decoders also have encoder-decoder attention layers.
- The encoder-decoder layers get the keys K and values V from the last self-attention layer of the encoder.

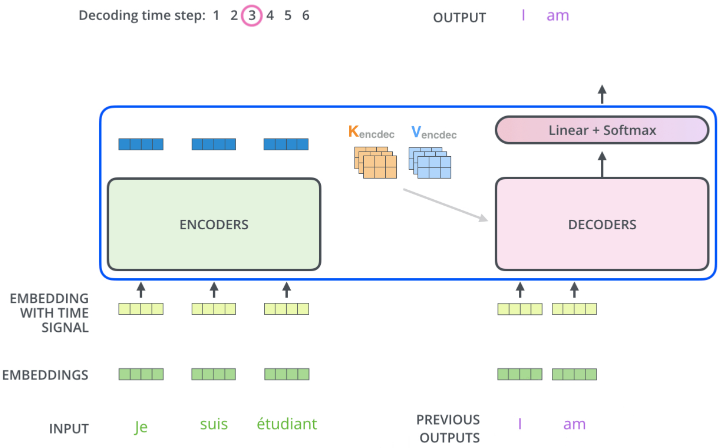


Autoregressive Inference of the Transformer



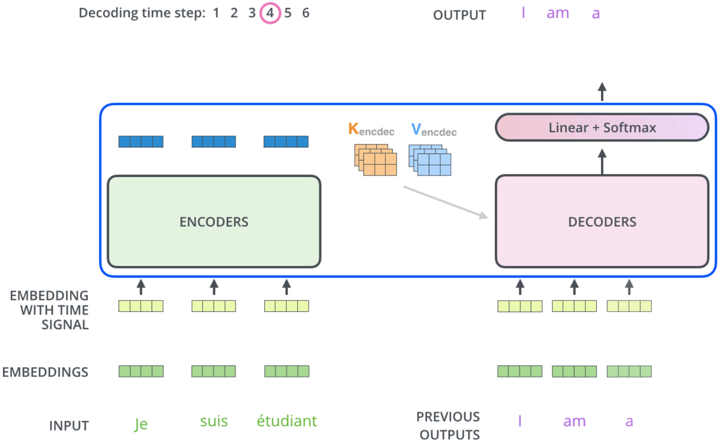
[Image source: https://jalammar.github.io/illustrated-transformer/]

Autoregressive Inference of the Transformer



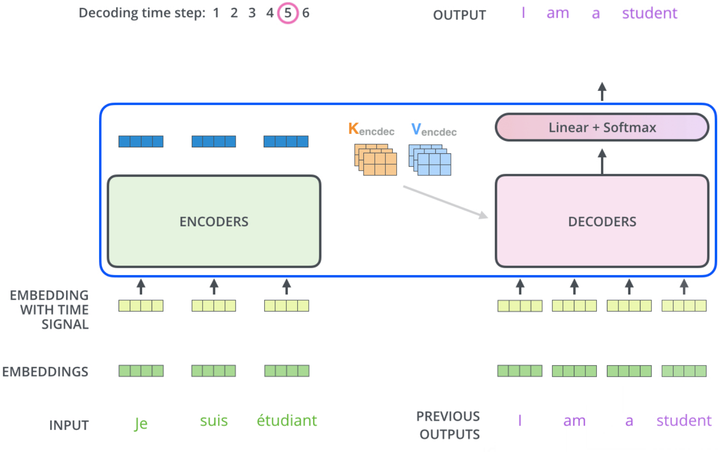
[Image source: https://jalammar.github.io/illustrated-transformer/]

Autoregressive Inference of the Transformer



[Image source: https://jalammar.github.io/illustrated-transformer/]

Autoregressive Inference of the Transformer



[Image source: `https://jalammar.github.io/illustrated-transformer/`]

Overview

1 Transformer

2 Attention Masks

3 Embedding

4 Applications

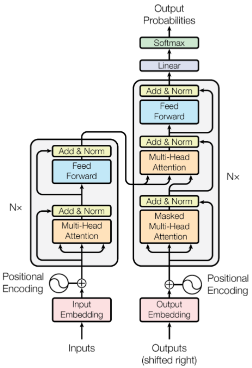
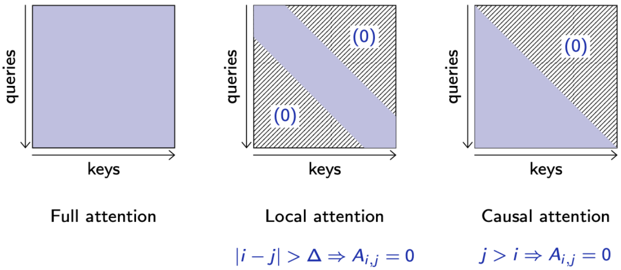


Figure: Original Transformer

[Image source: Vaswani et al. 2017]

Attention Masks in Transformers



- Full attention: Quadratic complexity $\mathcal{O}(n^2)$, used in Encoder architecture
- Local attention: Linear complexity $\mathcal{O}(n)$
- Causal attention: Quadratic complexity $\mathcal{O}(n^2)$, used in Decoder architecture:
  - Different to RNNs, this allows the Decoder to train on a whole sentence in parallel.
  - Predictions can only access past information preventing attention to future parts.

[Image source: `https://fleuret.org/dlc/`]

Overview

1 Transformer

2 Attention Masks

3 Embedding

4 Applications

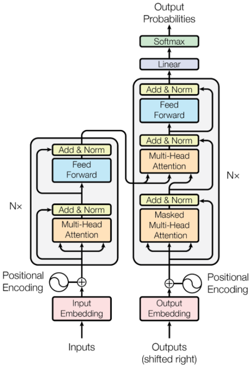


Figure: Original Transformer

[Image source: Vaswani et al. 2017]

Self-Attention is Permutation Invariant

- Given a permutation σ and a matrix $B \in \mathbb{R}^{d \times d}$, we will use the following notation for the permutation of the rows $\sigma(B)_i = B_{\sigma(i)}$.
- Remember the formula for attention was defined as

$$\text{attention}(q, K, V) = \textstyle\sum_{i=1}^{N} \text{similarity}(q, k_i)\, v_i.$$

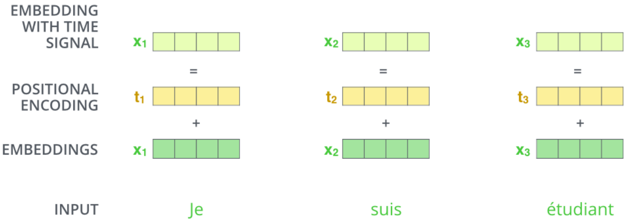
- Due to summing up, it directly follows that the standard attention operation is permutation invariant regarding K and V:

$$\text{attention}(q, \sigma(K), \sigma(V)) = \text{attention}(q, K, V)$$

- Thus, in general the attention can not see whether a word is the first one in a sentence!

Adding positional Embeddings

- Add positional information into the embedding vector.
- This information can then be used by the query and key matrices.



How to embed words?

```python
word_to_ix = {"hello": 0, "world": 1}
embeds = nn.Embedding(2, 5)  # 2 words in vocab, 5 dimensional embeddings
lookup_tensor = torch.tensor([word_to_ix["hello"]], dtype=torch.long)
hello_embed = embeds(lookup_tensor)
print(hello_embed)
```

Out:

```
tensor([[ 0.6614,  0.2669,  0.0617,  0.6213, -0.4519]],
       grad_fn=<EmbeddingBackward0>)
```

Figure: For each possible integer value a vector is assigned.

- Each possible word or token gets an embedding vector x assigned.
- The embedding vectors are also optimized via backpropagation.

[Image source: PyTorch Embeddings Tutorial]

Overview

1 Transformer

2 Attention Masks

3 Embedding

4 Applications

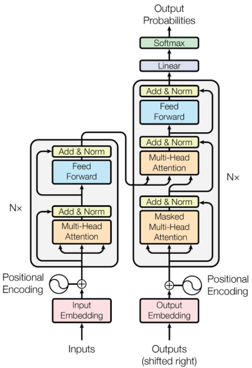


Figure: Original Transformer

[Image source: Vaswani et al. 2017]

Generative Pre-Trained Transformers (GPT)

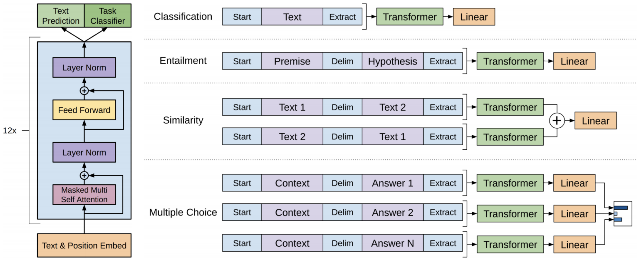


Figure: The GPT architecture consists out of the decoder part with slightly different skip connections.

- Simple architecture used for GPT1, GPT2, GPT3 and ChatGPT.
- The exercise is based on the GPT architecture.

Vision Transformer

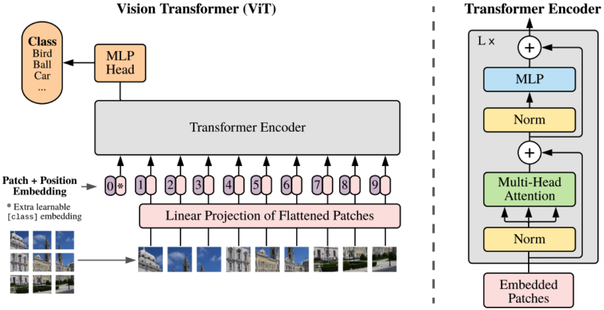


Figure: The vision transformer splits an image into patches that are handled as a sequence.

- The continuous input variables are only preprocessed by a linear projection.

[Image source: Dosovitskiy et al. 2021]

Sequential decision-making: Trajectory Transformer

- With Transformer we can learn a world model that allows us to optimal decision making.
- The Trajectory Transformer treats Reinforcement Learning as a single sequence problem.

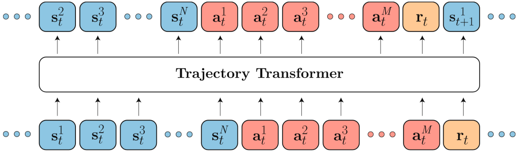


Figure: The Trajectory Transformer treats each dimension of states and actions separately and discretizes the state and action space.

- Optimization is done by conditioning on returns (the reward-to-go becomes an additional input in the sequence).
- We then can ask the Trajectory Transformer to create state-action-reward trajectories that have a high return.

[Janner et al. 2021]

Generalist agents: RT-2

- Transformer can flexibly handle multiple modalities like images, natural language or control signals at the same time.
- In robotics, Large Language Models (LLM) are now used regularly to encode task description and allow to have a better understanding of the environment.

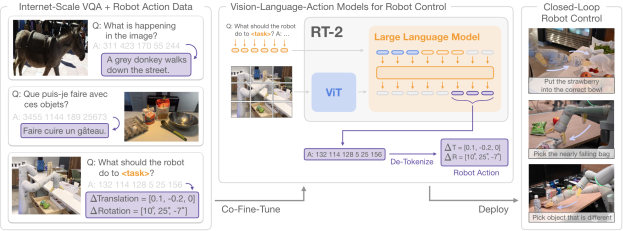


Figure: RT-2 is a novel vision-language-action (VLA) model that learns from both web and robotics data and translates this knowledge into generalised instructions for robotic control.

[Image source: Brohan et al. 2023]

Some Advantages and Disadvantages of Transformers

Advantages:

- **State-of-the-Art Performance:** Transformers have achieved remarkable results in natural language processing as well as computer vision.
- **Long-Range Dependencies:** They handle long-range dependencies well, making them suitable for tasks that require capturing global context.
- **Multi-Modality Fusion:** Transformers excel at fusing information from different modalities like images, language or time series data, making them versatile in handling multi-modal data.

Drawbacks:

- **Computational Complexity:** Transformers can be computationally expensive and memory-intensive, limiting their scalability.
- **Data Requirements:** Training Transformers often requires large datasets as they have less inductive bias.

Further Material

- The Illustrated Transformer (a lot of visualizations in this lecture are based on it):
  `https://jalammar.github.io/illustrated-transformer/`
- Minimal GPT3 implementation (our exercise is based on this implementation):
  `https://github.com/karpathy/minGPT`
- A clear algorithmic description for Transformers:
  `https://arxiv.org/abs/2207.09238`