GenAI / Transformers Workshop
Overview, Internals and Insights

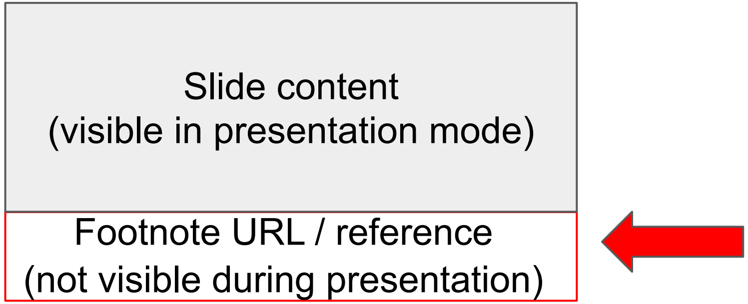
DKRZ
DEUTSCHES
KLIMARECHENZENTRUM

HELMHOLTZ AI | ARTIFICIAL INTELLIGENCE
COOPERATION UNIT

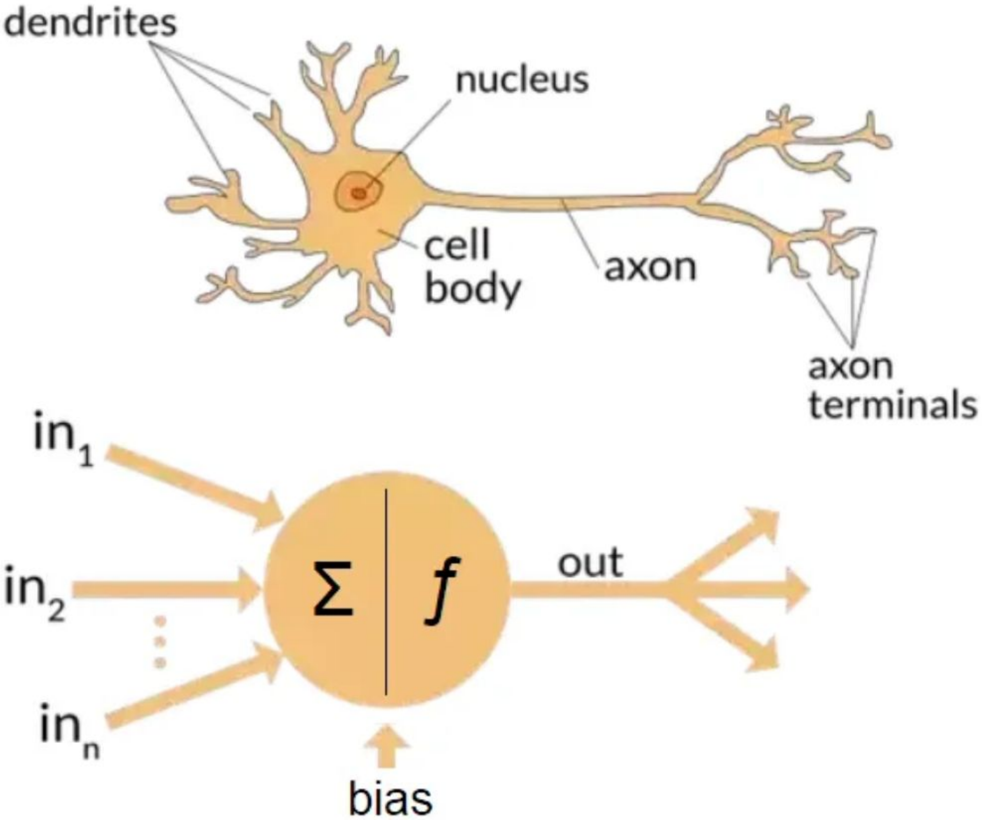
Caus Danu

Background

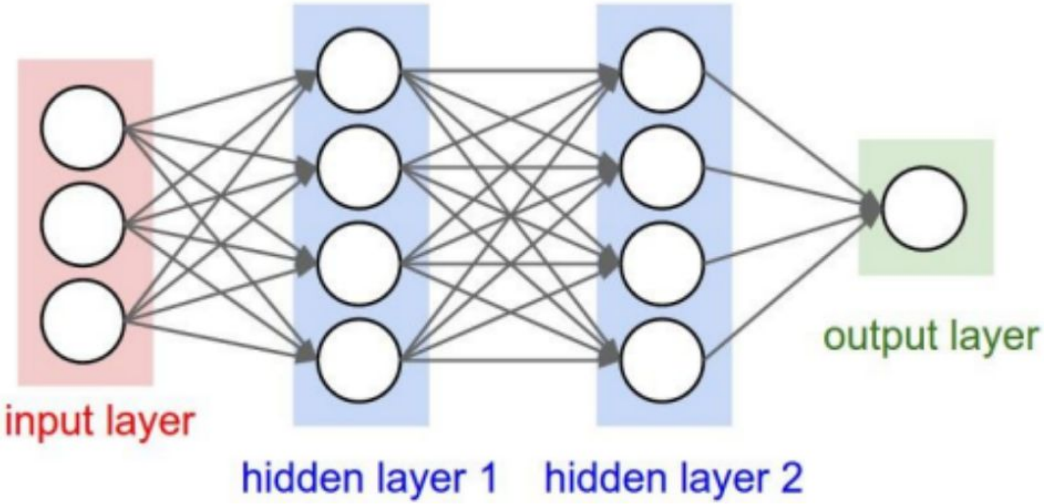
- This resource is intended to introduce neural networks, generative AI at a high abstract level and subsequently focus on **explaining Transformers** in more depth to a scientific audience.
- Transformers will be presented in the broader context of AI, building the stage from zero, step by step.
- I **will act** therefore **as a synthesizer** of many resources created by the broader AI community.
- Hence, **many thanks!** to all the creators of the helper material. All credits and references are visible on the last slide.
- Same references are also given in the **footnotes section** of each individual slide, which is **not visible in presentation mode**, but will be useful for referencing at home for extra study if need be.



Artificial Neural Nets and Brain Parallels

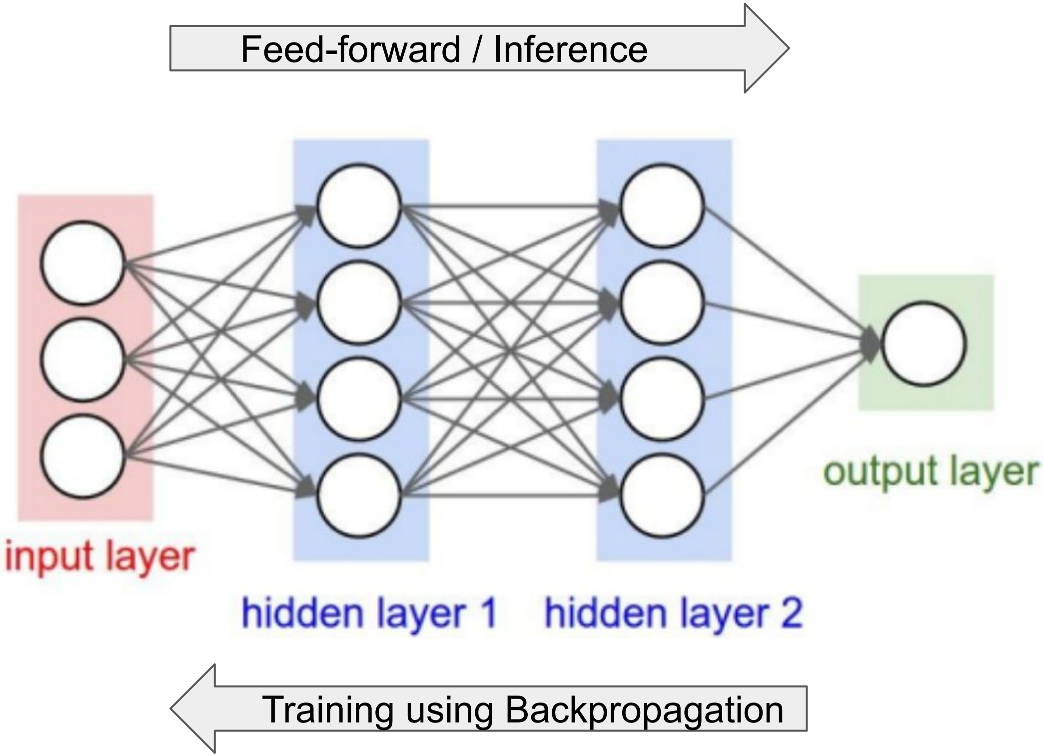


Artificial Neural Networks (ANNs)



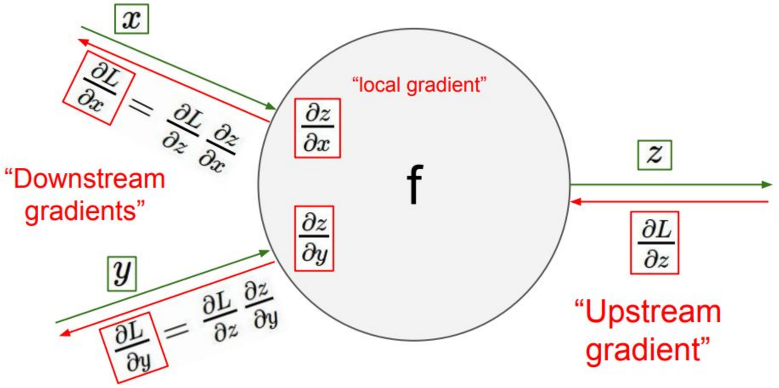
CS231n course from Stanford University

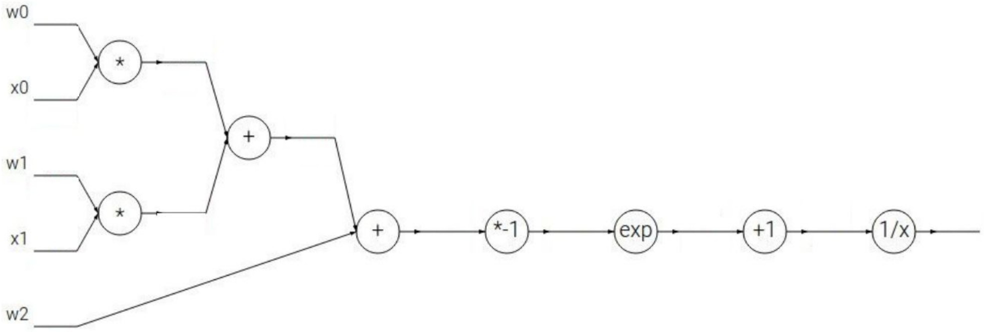
Backpropagation



CS231n course from Stanford University

Backpropagation using Local Gradients and Chain Rule

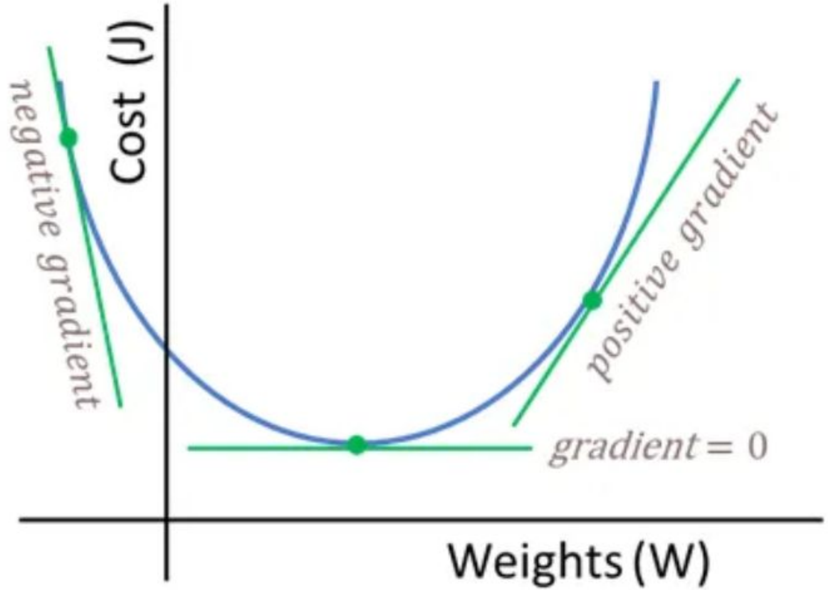




- ➢ The Chain Rule is implemented with the help of **local gradients**.
- ➢ We **recursively multiply** the local derivatives.
- ➢ Backpropagation is a **recursive** application of the chain rule backwards through the **computation graph**.

CS231n course from Stanford University

Cost function



Literature naming conventions: Cost / Loss / Error function

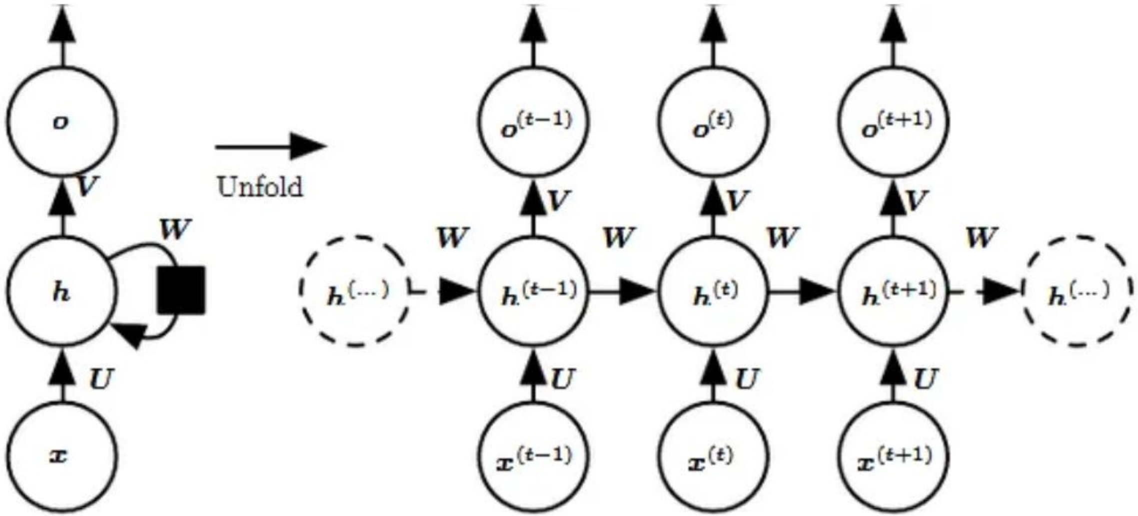
Weights update

$$W_{new} = W_{old} - \alpha \underbrace{\frac{dJ}{dW}}_{\text{gradient}}$$

- **J** and **L** are usual notations for the **Loss** / **Error** / **Cost** function, i.e. the difference between what the model **predicts** and what it should predict according to the **ground truth**.

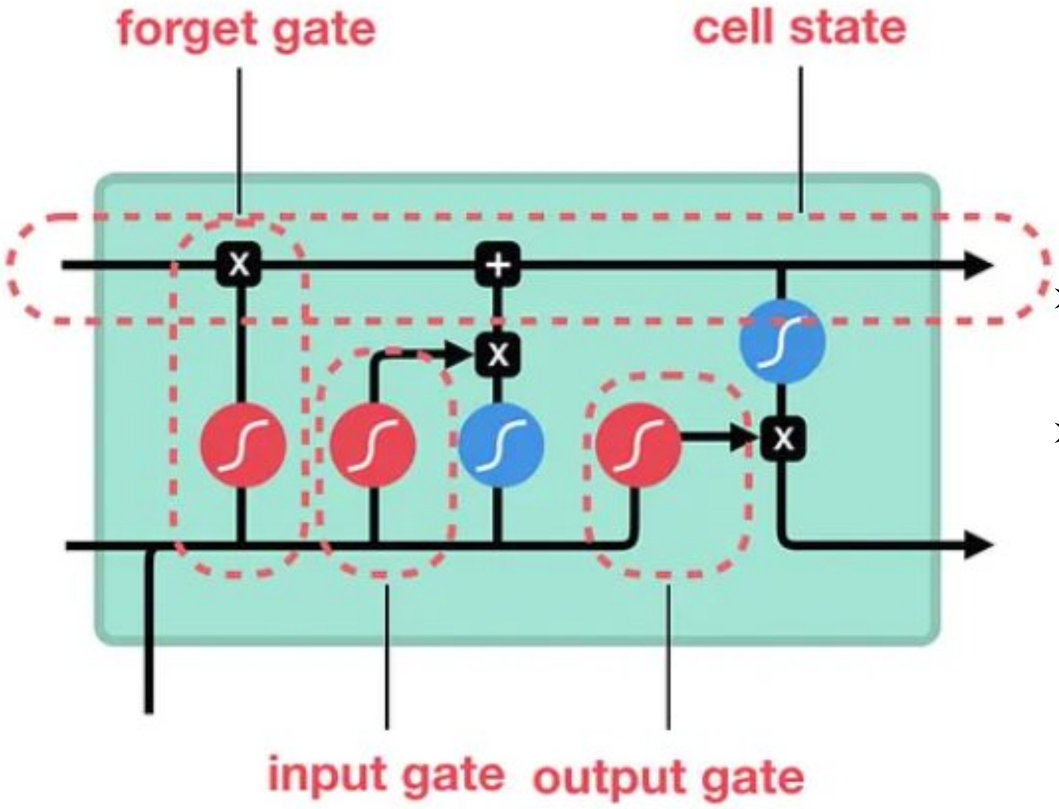
- The **weights are updated** in the direction of the **negative** gradient, so that the **cost function is minimized** as much as possible.

Sequential nature of Recurrent Neural Networks (RNNs)



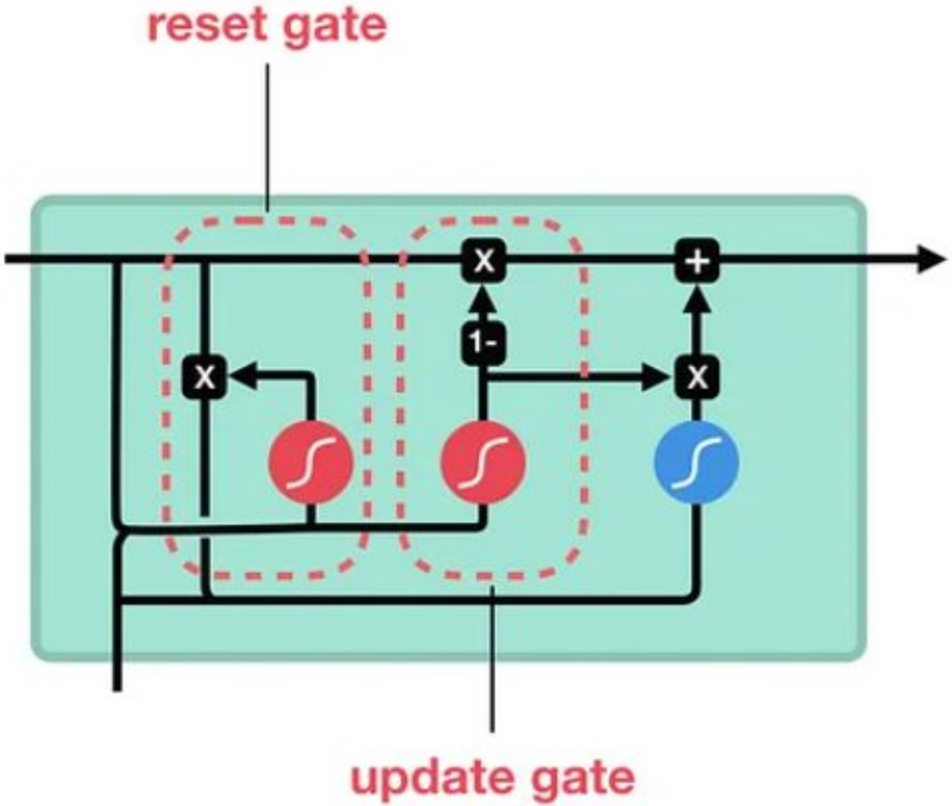
- By **unfolding** the feedback loop in time, we become aware of the complexity of these networks. It is as if we train a **very deep network** and that is why they are harder to train.
- With **RNNs** things are done **sequentially** => **deep** graph structure.
- With **Transformers** things happen **in parallel** => **broad** graph structure.
- **Transformers** might simply be **easier to train stably**, and maybe that is why they have better results.

Long short-term memory (LSTMs)



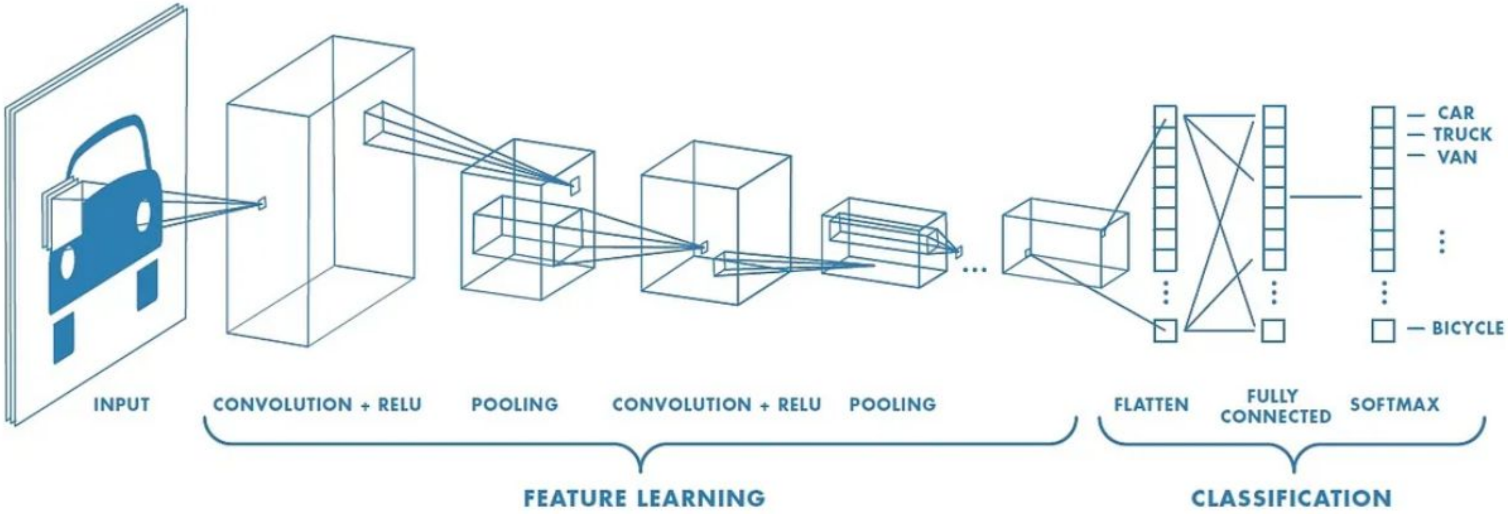
- ➢ Contains special **gates** that address the problem of **vanishing gradients**.
- ➢ Addresses the problem of **exploding gradients**.

Gated Recurrent Units (GRUs)



- ➢ Generally considered **faster** than LSTMs.
- ➢ In practice => **similar outcomes to** what **LSTM** provides.

Convolutional Neural Networks (CNNs)



- Generally applied in **computer vision** tasks, i.e. **2D image focused, not time sequence data**.
- We can use **3D CNNs to handle sequences of data**, where 3rd dimension is time. Here we talk about a cube kernel, instead of a plane 2D kernel.
- CNNs are very amenable to **parallelism**.

Modeling Spectrum in the current AI Era

Discriminative models
[older established approach]

Generative models
[newer trend]

Best of both worlds predictions => Inductive bias + Data

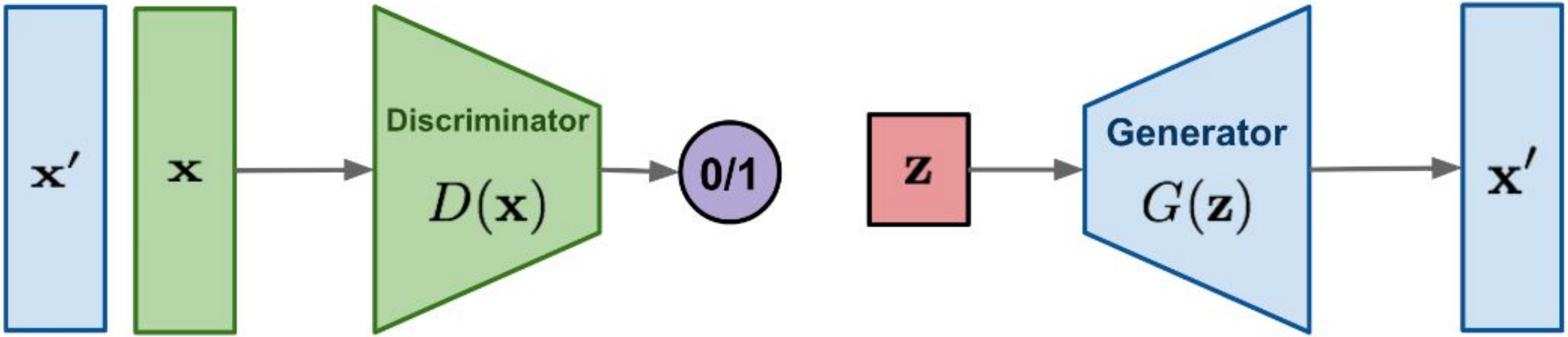
➢ Use **a lot of training data** / throw a lot of data at the problem task and **let the network figure it out**
➢ Predictions are limited to the training data domain, i.e. tied to the train dataset statistics, hence **poor out of distribution predictions**

➢ Is capable of **out of domain generalization**
➢ Lots of inductive bias, **a priori knowledge** is injected / enforced as a guide during training
➢ **More creative** than discriminative models

Main Generative AI models

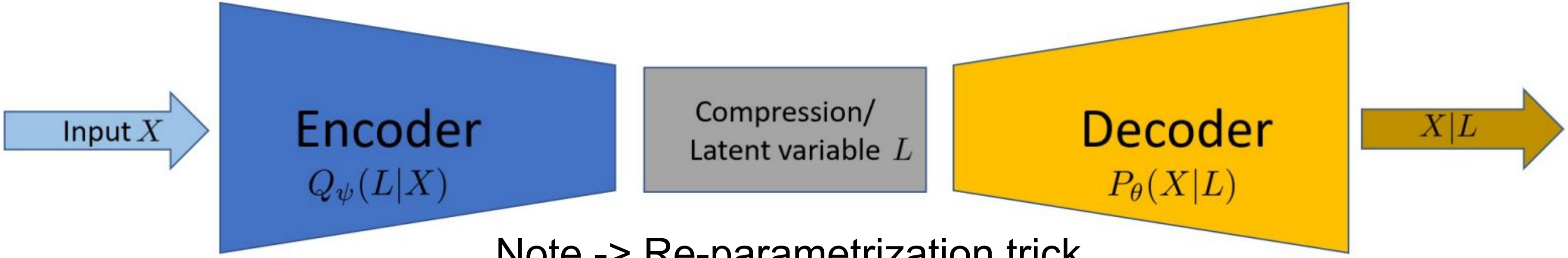
- Generative Adversarial Networks / GANs -> **Adversarial** training / **Arms race**
- Variational Autoencoders / VAEs -> Autoencoder, but with **latent space**
- Flow-based models -> **Invertible mapping** between distributions
- Diffusion models -> **Markov chain** for **denoising** intermediate states
- **Transformers -> we will focus on them in this workshop**

Generative Adversarial Networks / GANs



- Two networks: Generator and Discriminator play a **min-max game**
- **Generator** aims at producing realistic outputs **to trick the Discriminator**
- **Discriminator** strives to improve its ability to **discern true from fake** data

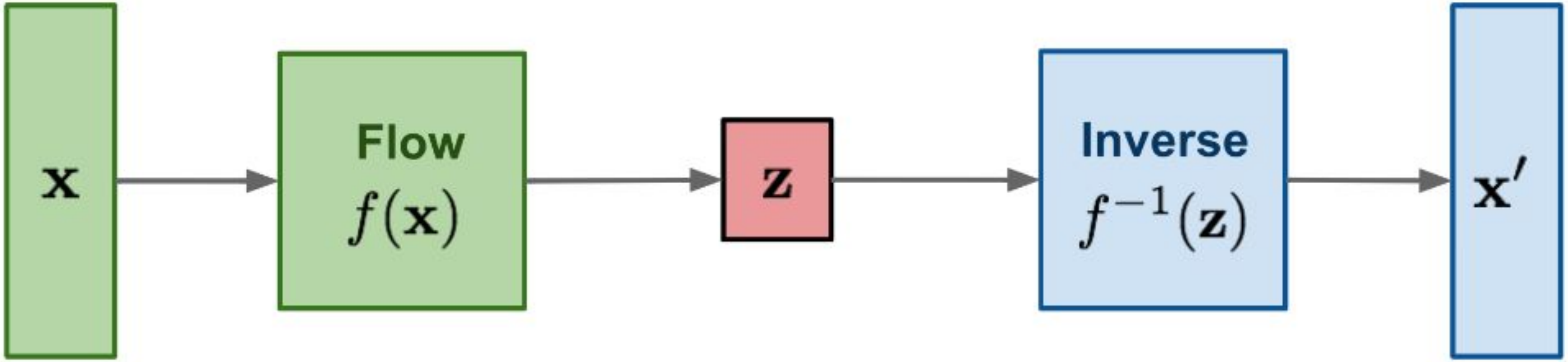
Variational Autoencoders / VAEs



Note -> Re-parametrization trick

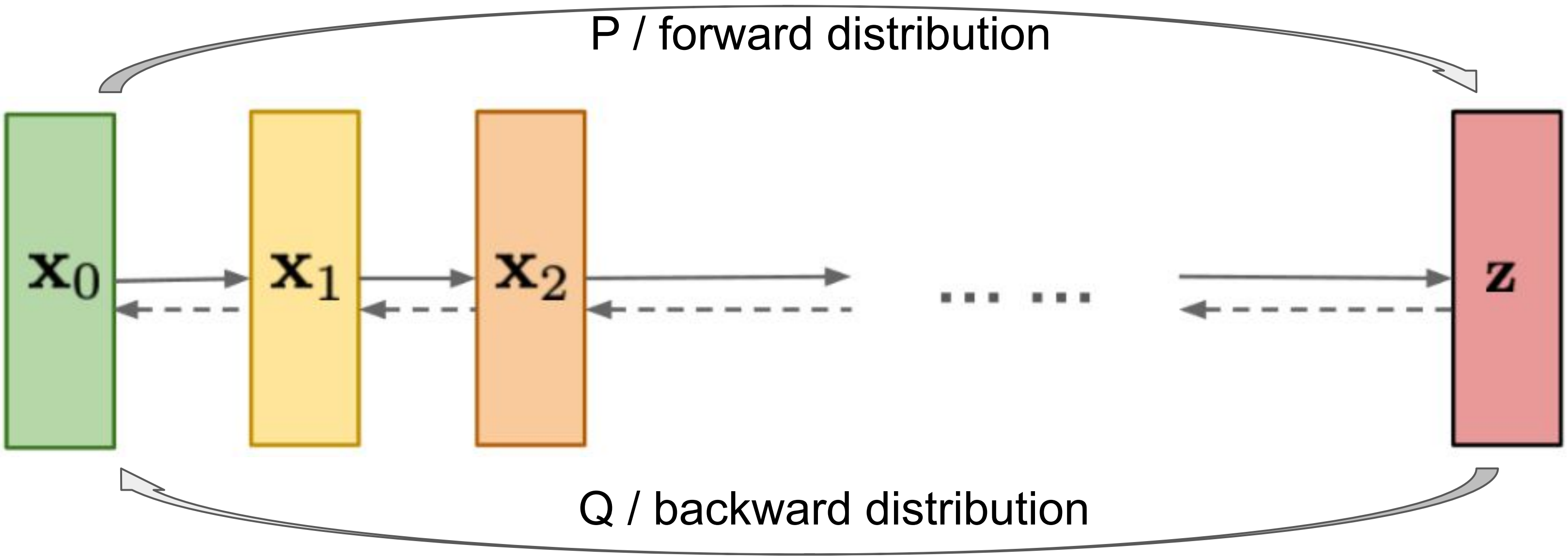
- Inspired from traditional **autoencoders** that **compress** data into **vector codes**
- Variational autoencoders will learn a **latent distribution function** instead
- The **latent distribution can be sampled**, resulting in variable latent samples
- The samples are **decoded**, resulting in a **variety of realistic reconstructions**

Flow-based models



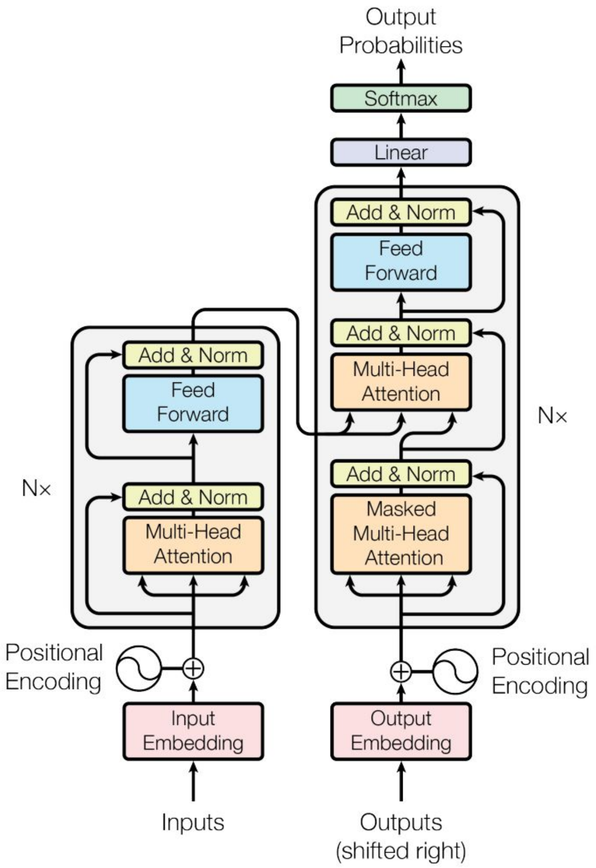
- ➢ Use **invertible neural networks** to learn the data distribution
- ➢ Enable **exact log-likelihood** directly thanks to one-to-one invertibility
- ➢ **More** computationally **expensive than VAEs**, due to invertibility requirements

Diffusion models



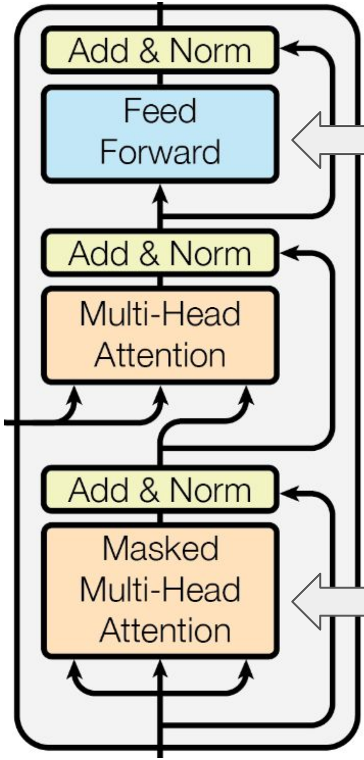
- **Markov chain** of states, where each state is a noisy version of the other
- The original input X_0 is **diffused into** pure **noise Z** over several steps
- The model **learns the noise that needs to be removed from a state** at time **t**, to get to an **earlier state**

Transformers



“Attention is all you need” paper from 2017 by Vaswani et al.

Transformer Block



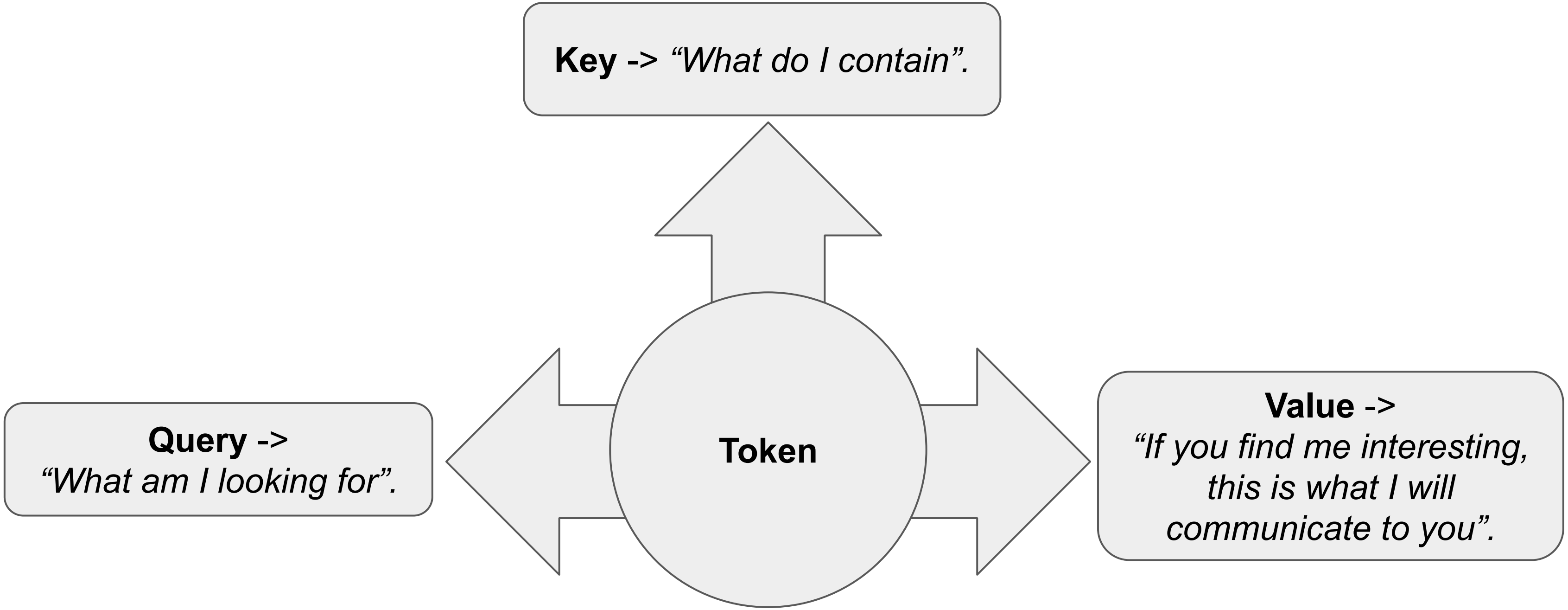
2. Secondly, a **Multilayer Perceptron (MLP)** crunches the data that was communicated amongst tokens in the previous self-attention phase.

1. First the tokens **communicate** between themselves / they attend to each other (**self-attention**).

Self-Attention

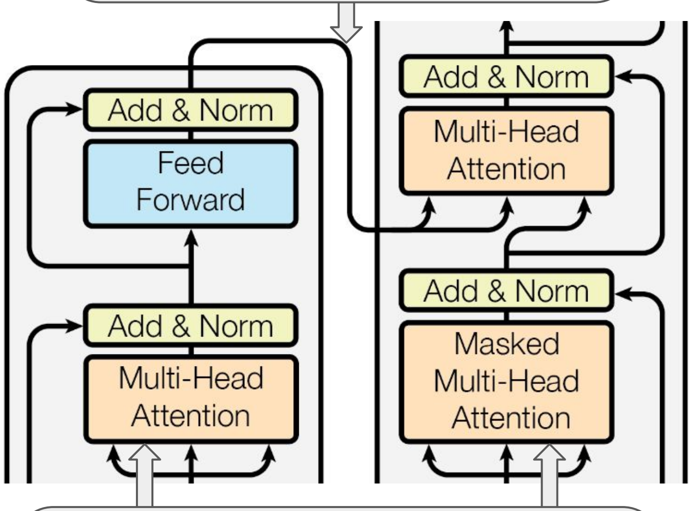
- ➢ **All tokens communicate with one another**.
- ➢ This is computationally expensive because **each token has to look at every other token** to compute an **attention score / attention weight**.

Key, Query and Value Embeddings



Self-Attention vs Cross-Attention

Cross-Attention -> the *queries* come from the **decoder**, whereas the *keys* and *values* are from the **encoder** side.



Self-attention -> the *key*, *query* and *value* vectors are related to the **same entity**, either the encoder, or the decoder.

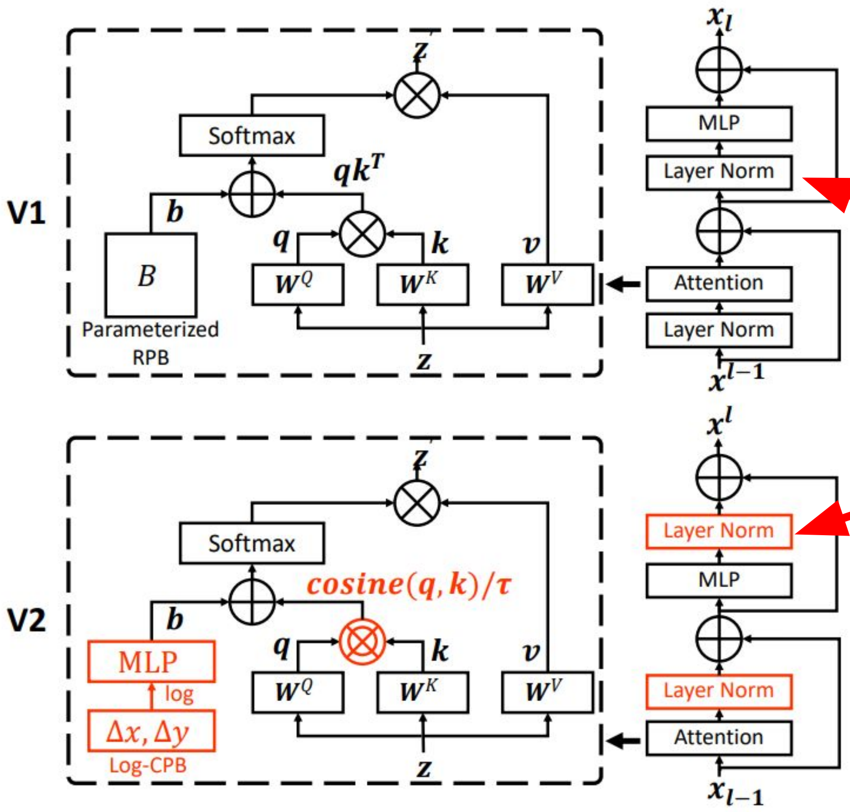
Mathematically Expressing Self-Attention

- ➢ The dot product **Query ▪ Key** is the attention score, where **Query** and **Key** are **embedding vectors**.

- ➢ Dot product measures **similarity between vectors** => **Attention** can be interpreted as the **alignment** between the **Key** and the **Query** vectors (i.e. two tokens find each other interesting).

- ➢ Instead of the dot product, other measures can be used, like the **cosine similarity** for example (**Swin Transformer Version 2** paper).

Cosine similarity



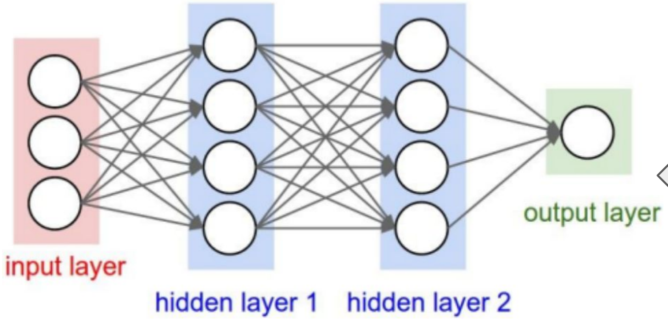
Notice the **pre**-Layer Norm
vs
post-Layer Norm
switch.

"Swin Transformer V2: Scaling Up Capacity and Resolution" by Liu et al.

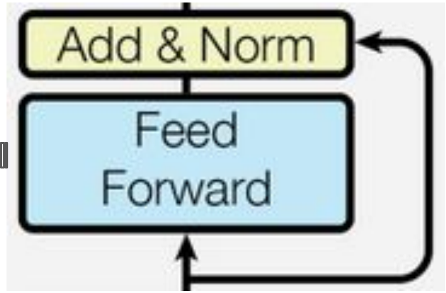
Computation Phase / Feed-Forward MLP

- After the communication between tokens is finished, an MLP has to **"think"** on what was **"said"** during the self-attention phase.

- This basically means that new features are computed / derived as a result of the communication.



Let's fast forward back to what we've already seen



Positional encoding

- ➢ The transformer treats the tokens as a **Bag of Words (BoW)**.
- ➢ We need to give each token a label that specifies its position in the form of a **counter ID** for instance.
- ➢ It is interesting that the positional encoding information is **literally added** by a **"+"**/ **plus** operation.

*There are various encoding schemes such as for example **absolute encoding**, **relative encoding**, that have a significant impact on how the transformer performs in the end. Check out the Swin Transformer paper for empirical proof.*

ChatGPT Pipeline

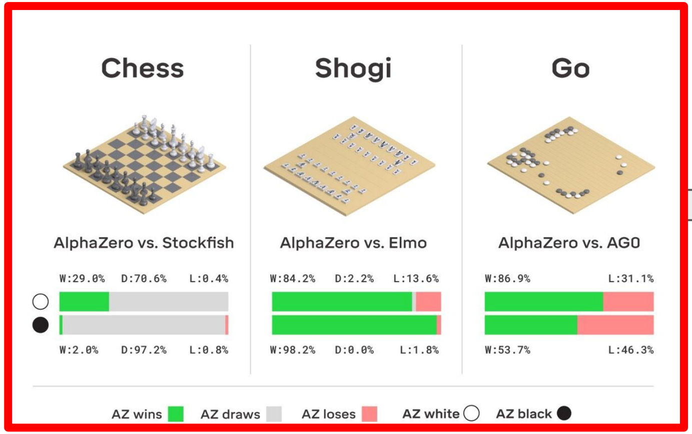
1) Pretraining the **base model**.

2) Supervised Fine-Tuning (**SFT**).

3) **Reward** Modeling / **RM**.

4) Reinforcement Learning / **RL** (Very much research territory at the moment).

Personal opinion: ChatGPT works so well, because it borrowed many insights from the AlphaZero games playing engine from back in 2016. *Words are the new chess pieces that have to be smartly arranged.*



DeepMind subsequently created **AlphaStar** and **AlphaFold** using similar principles.

Pretraining the Foundational Model

- ➢ Use **raw data** to train a **Base Model**.
- ➢ The dataset is **huge** => potentially low quality in some places, but **very large quantity**.
- ➢ We obtain a **document completer** in the end.
- ➢ **Thousands of GPUs** work in parallel (ex: 1000-2000 A100 GPUs).

Supervised Fine Tuning (SFT stage)

➢ **Low quantity, but very high quality data**: ~ 100K (prompt, response) tuples.

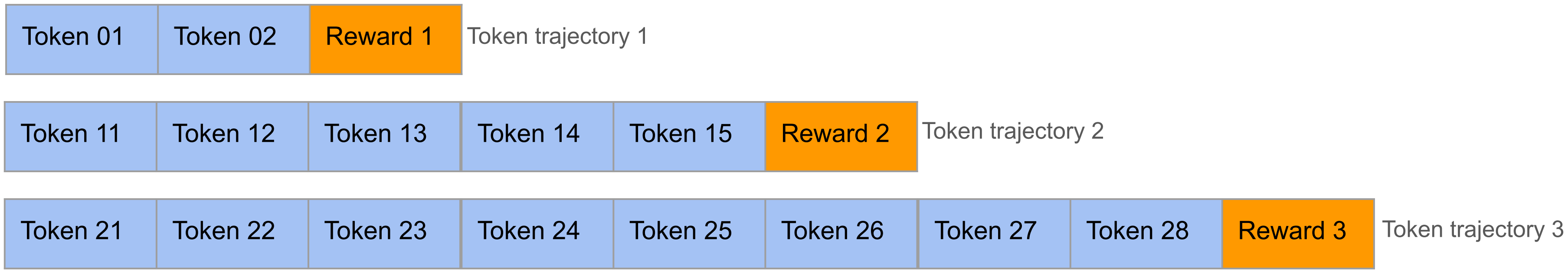
➢ Domain Specialists / Contractors have to scrutinize the dataset so that close to ideal **(prompt, response) tuples** are assembled.

➢ Less GPUs are required than in step 1 (ex: 1-100).

➢ The outcome is the so-called **SFT model**.

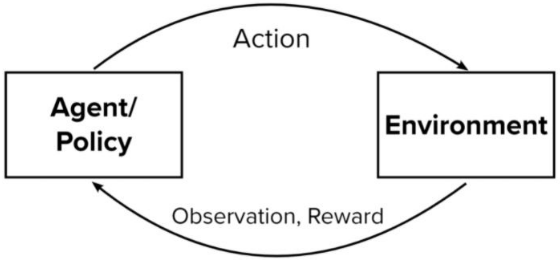
Reward Modeling

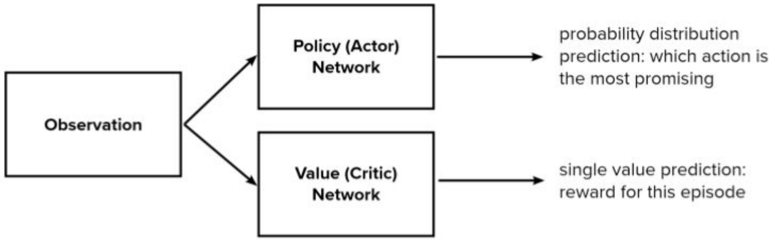
- Ask the SFT model to produce **multiple answers** per prompt.
- Ask contractors to carefully **rank these answers**.
- Train a reward model on these rankings.
- Order of 1 to 100 GPUs for training.
- The outcome is the so-called **Reward Model / RM** => **Evaluates token trajectories.**



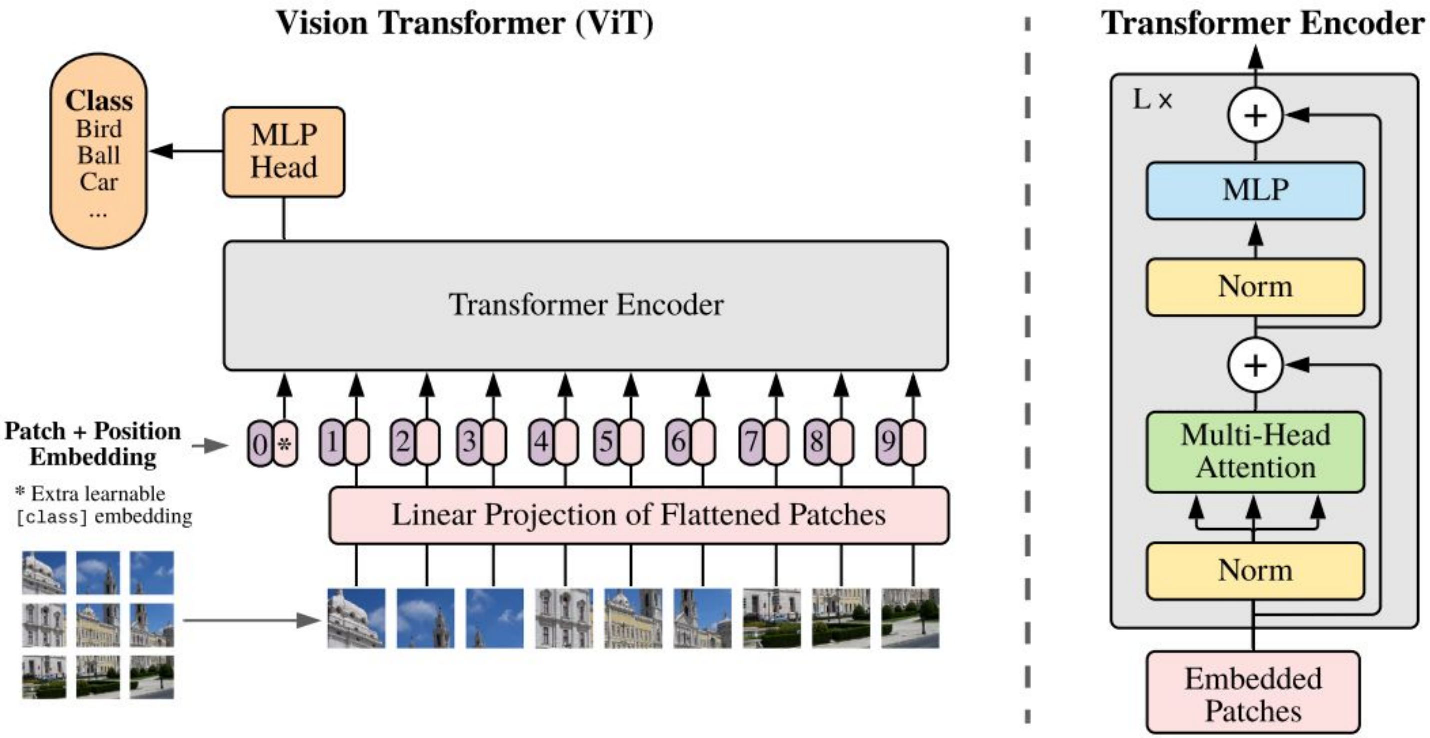
Reinforcement Learning

- ➢ Train a **PPO** algorithm (**Proximal Policy Optimization**).
- ➢ Use the previously trained reward model to **evaluate the reward**.
- ➢ PPO will have the task of generating **token "trajectories"** that will have a **very good overall score**.
- ➢ Order of 1 to 100 GPUs for training.
- ➢ The outcome is the so called **RL model / RLHF** (reinforcement learning with **human feedback**).



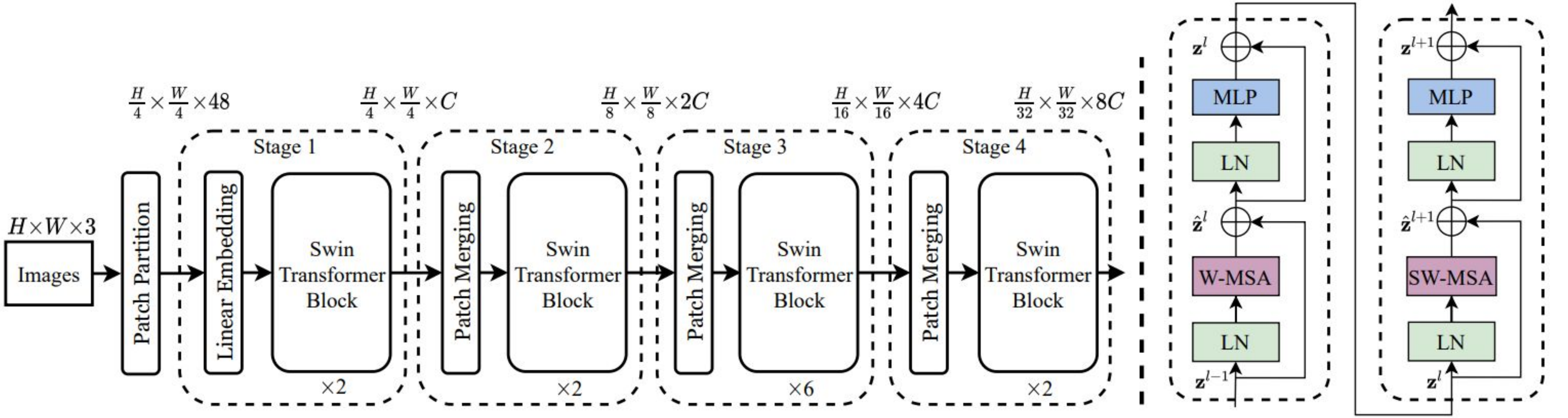


Vision Transformer / ViT



"An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale" by A. Dosovitskiy et al. (2021)

Swin Transformer



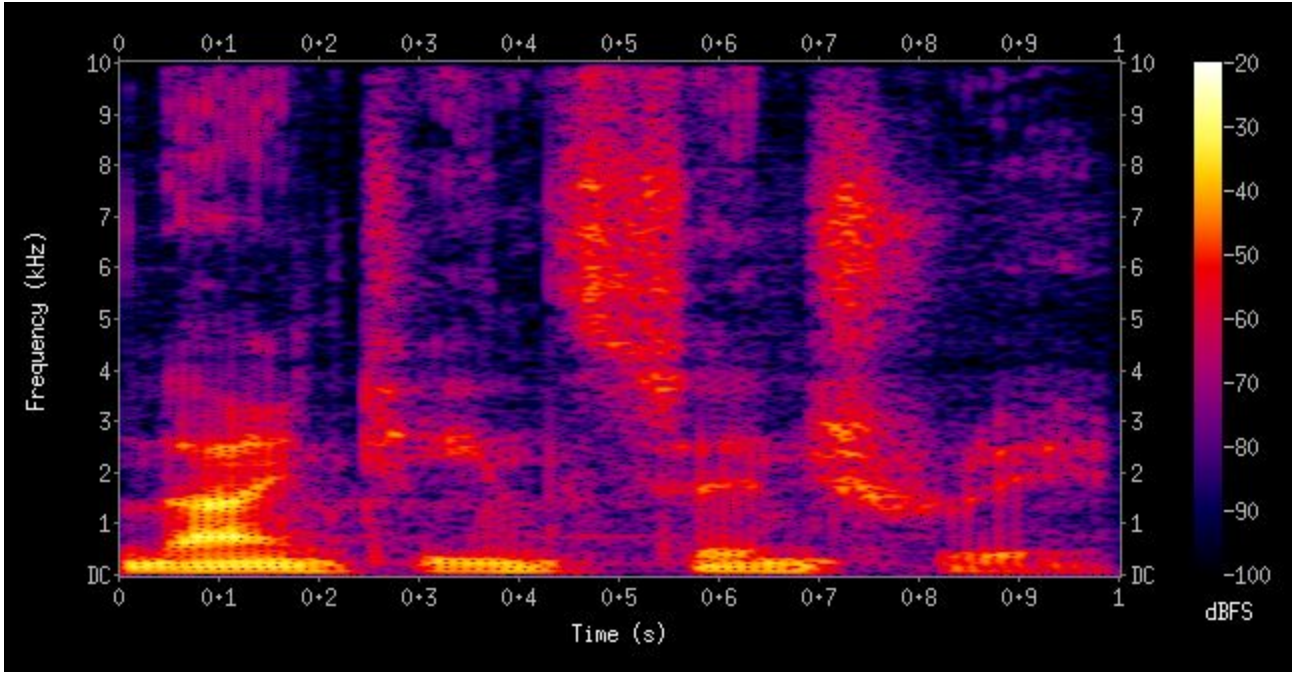
(a) Architecture

(b) Two Successive Swin Transformer Blocks

“Swin Transformer: Hierarchical Vision Transformer using Shifted Windows” by Liu et al. (2021)

Audio Transformer

Raw sound waves can be mapped to a different space using **STFT** (Short-time Fourier Transform).



Here time series become images => **problem is adapted** to be tackled by the ViT / Swin Transformer.

Time Series Transformer (TST)

Continuous data is **sampled** and **quantized** into **discrete tokens**.

Implementations available at: https://huggingface.co/docs/transformers/model_doc/time_series_transformer

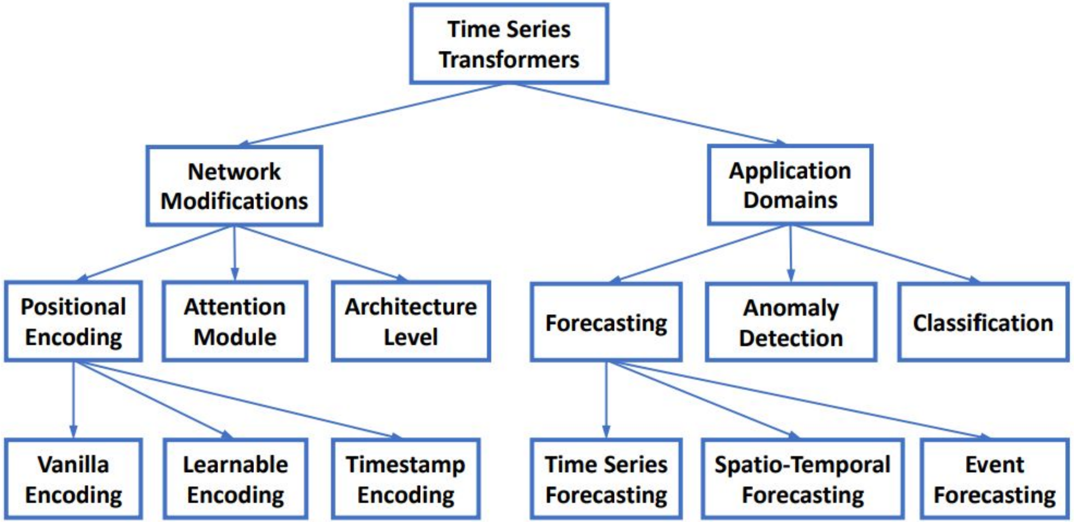


Figure 1: Taxonomy of Transformers for time series modeling from the perspectives of network modifications and application domains.

"Transformers in Time Series: A Survey" paper, by Wen et al. [2023]

Code example: Self-Attention Snippet Version 1

```python
# Version 1:
# We want x[b,t] = mean_{i<=t} x[b,i]
xbow = torch.zeros((B,T,C))
for b in range(B):
    for t in range(T):
        xprev = x[b,:t+1] # (t,C)
        xbow[b,t] = torch.mean(xprev, 0)

print(x[0])
print(xbow[0])
```

```
torch.Size([4, 8, 2])
tensor([[ 1.9269,  1.4873],
        [ 0.9007, -2.1055],
        [ 0.6784, -1.2345],
        [-0.0431, -1.6047],
        [-0.7521,  1.6487],
        [-0.3925, -1.4036],
        [-0.7279, -0.5594],
        [-0.7688,  0.7624]])
tensor([[ 1.9269,  1.4873],
        [ 1.4138, -0.3091],
        [ 1.1687, -0.6176],
        [ 0.8657, -0.8644],
        [ 0.5422, -0.3617],
        [ 0.3864, -0.5354],
        [ 0.2272, -0.5388],
        [ 0.1027, -0.3762]])
```

Code example: Self-Attention Snippet Version 2

```python
# Version 1
# We want x[b,t] = mean_{i<=t} x[b,i]
xbow = torch.zeros((B,T,C))
for b in range(B):
    for t in range(T):
        xprev = x[b,:t+1] # (t,C)
        xbow[b,t] = torch.mean(xprev, 0)
print(x[0])
print(xbow[0])

# Version 2
wei = torch.tril(torch.ones(T, T))
wei = wei / wei.sum(1, keepdim=True)
xbow2 = wei @ x # (B, T, T) @ (B, T, C) --->
(B, T, C)
print("Are xbow and xbow2 the same? -> ",
torch.allclose(xbow, xbow2))
```

```
torch.Size([4, 8, 2])
tensor([[ 1.9269,  1.4873],
        [ 0.9007, -2.1055],
        [ 0.6784, -1.2345],
        [-0.0431, -1.6047],
        [-0.7521,  1.6487],
        [-0.3925, -1.4036],
        [-0.7279, -0.5594],
        [-0.7688,  0.7624]])
tensor([[ 1.9269,  1.4873],
        [ 1.4138, -0.3091],
        [ 1.1687, -0.6176],
        [ 0.8657, -0.8644],
        [ 0.5422, -0.3617],
        [ 0.3864, -0.5354],
        [ 0.2272, -0.5388],
        [ 0.1027, -0.3762]])
Are xbow and xbow2 the same? ->  True
```

Code example: Self-Attention Snippet Version 3

```
# Version 2
wei = torch.tril(torch.ones(T, T))
wei = wei / wei.sum(1, keepdim=True)
xbow2 = wei @ x # (B, T, T) @ (B, T, C) --->
(B, T, C)
print("Are xbow and xbow2 the same? -> ",

# Version 3: using Softmax
tril = torch.tril(torch.ones(T,T))
wei = torch.zeros((T,T))
wei = wei.masked_fill(tril == 0,
float('-inf'))
wei = F.softmax(wei, dim=-1)
xbow3 = wei @ x
print("Are xbow/xbow2 equal to xbow3? -> ",
torch.allclose(xbow, xbow3))
```

```
torch.Size([4, 8, 2])
tensor([[ 1.9269,  1.4873],
        [ 0.9007, -2.1055],
        [ 0.6784, -1.2345],
        [-0.0431, -1.6047],
        [-0.7521,  1.6487],
        [-0.3925, -1.4036],
        [-0.7279, -0.5594],
        [-0.7688,  0.7624]])
tensor([[ 1.9269,  1.4873],
        [ 1.4138, -0.3091],
        [ 1.1687, -0.6176],
        [ 0.8657, -0.8644],
        [ 0.5422, -0.3617],
        [ 0.3864, -0.5354],
        [ 0.2272, -0.5388],
        [ 0.1027, -0.3762]])
Are xbow/xbow2 equal to xbow3? ->  True
```

Takeaways

- ➢ Generative AI is the **art of encoding complex real world distributions**, such that we can **generate creative results** later **via sampling** from the encoded distribution.

- ➢ Transformers are powerful neural networks that **borrow the best ideas** from prior models in the AI ecosystem and **combine them together for a synergistic effect**.

- ➢ **Self-attention** and **Feed-Forward MLP** are the major conceptual components of a Transformer **block**.

- ➢ **Self-attention** is essentially a **communication graph** where tokens exchange **information stored in channels** amongst themselves.

- ➢ The **Feed-Forward MLP** is used for the **computation phase to learn embeddings**. Better embeddings means better **abstract "meanings"** are learned in a high dimensional space, resulting in better predictions.

- ➢ **Residual connections** and **pre- / post-normalization** are other important attributes to help towards successful training and faster convergence.

References

- Slide 3: https://towardsdatascience.com/the-differences-between-artificial-and-biological-neural-networks-a8b46db828b7
- Slide 4, 5, 6: http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture04.pdf
- Slide 7, 8: https://towardsdatascience.com/neural-networks-backpropagation-by-dr-lihi-gur-arie-27be67d8fdce
- Slide 9: https://towardsdatascience.com/recurrent-neural-networks-rnns-3f06d7653a85
- Slide 10, 11: https://medium.com/towards-data-science/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21
- Slide 12: https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148
- Slide 15, 17: https://lilianweng.github.io/posts/2018-10-13-flow-models/
- Slide 16: https://ducspe.github.io/masterthesis_danucaus/ -> (page 18)
- Slide 18: https://lilianweng.github.io/posts/2021-07-11-diffusion-models/
- Slides 19, 20, 23, 26, 27 Transformer components were taken from “Attention is all you need” paper, by Vaswani et al.
- Slide 25: “Swin Transformer V2: Scaling Up Capacity and Resolution” paper, by Liu et al.
- Slide 28: https://arstechnica.com/science/2018/12/move-over-alphago-alphazero-taught-itself-to-play-three-different-games/
- Slide 32: https://odsc.com/blog/reinforcement-learning-with-ppo/
- Slide 33: “An image is worth 16x16 words: Transformers for image recognition at scale” paper, by Dosovitskiy et al.
- Slide 34: “Swin Transformer: Hierarchical Vision Transformer using Shifted Windows” paper, by Liu et al.
- Slide 35: https://en.wikipedia.org/wiki/Short-time_Fourier_transform#/media/File:Spectrogram-19thC.png
- Slide 36: “Transformers in Time Series: A Survey” paper, by Wen et al.