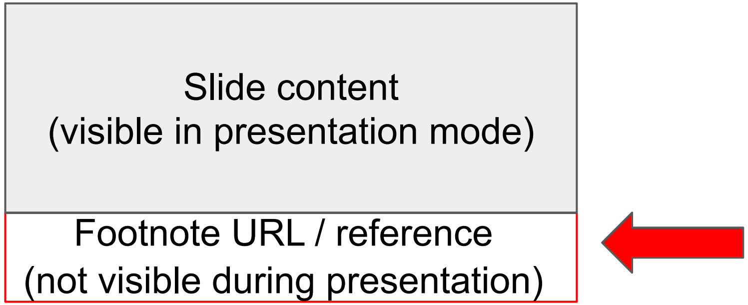
Transformers Workshop

Internals and Insights

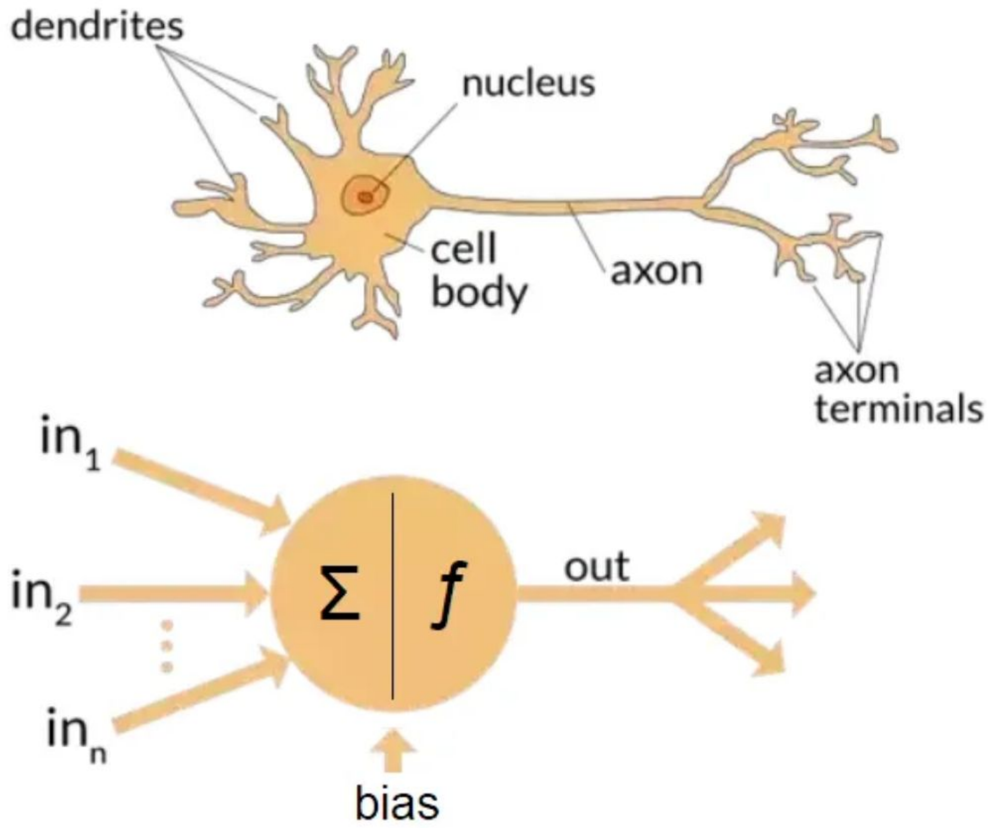
Caus Danu

Background

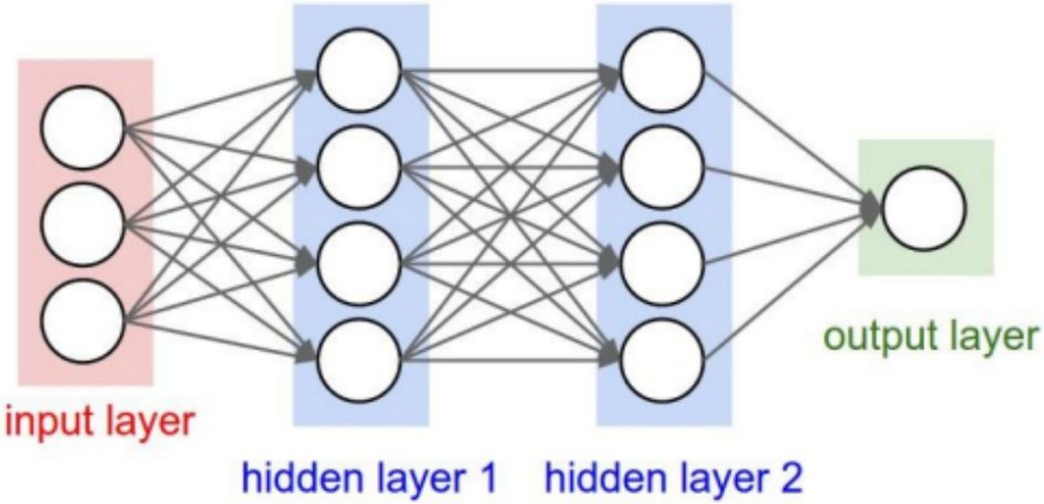
- This resource is intended to **explain Transformers** to a scientific audience.
- Transformers will be presented in the broader context of AI, so touching **different other relevant topics**.
- **I will act** therefore **as a synthesizer** of many resources created by the broader AI community.
- Hence, **Many Thanks!** to all the creators of the helper material. All credits and references are visible on the last slides.
- 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

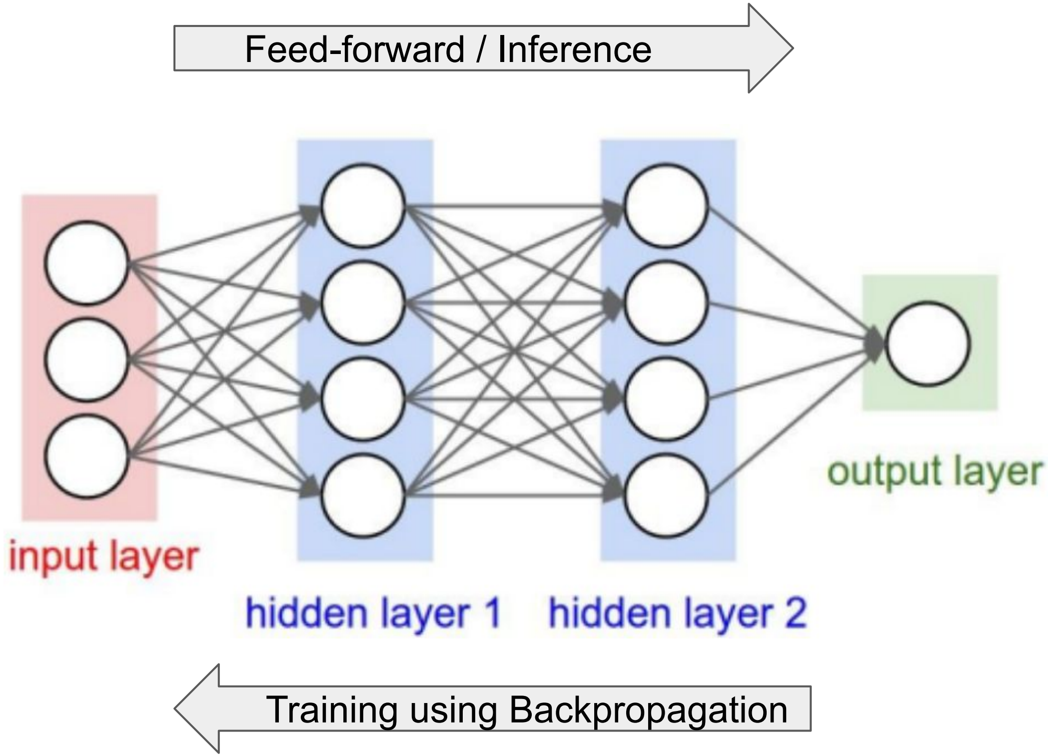


Artificial Neural Networks (ANNs)



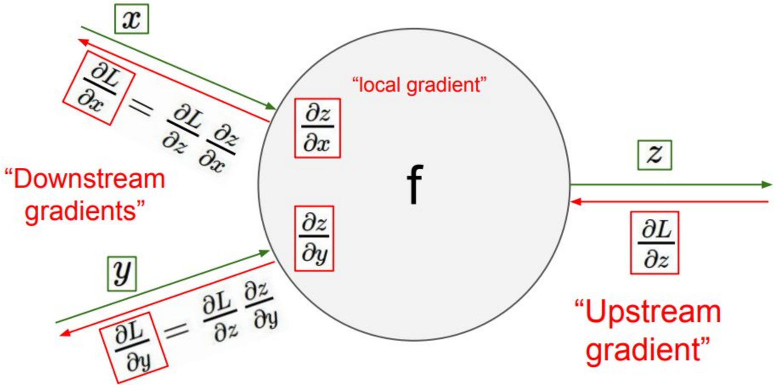
CS231n course from Stanford University

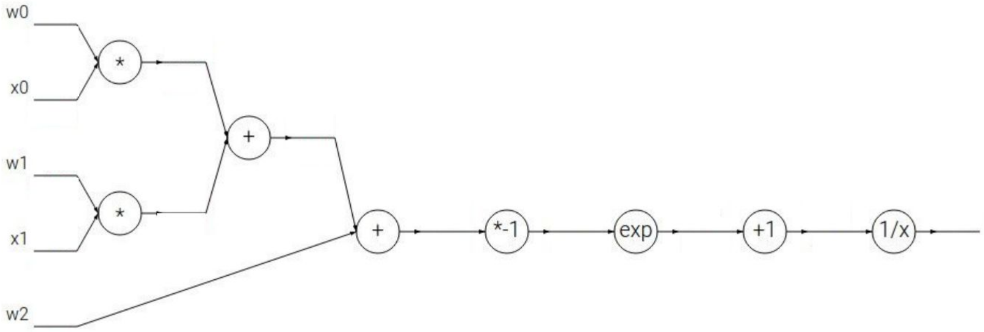
Backpropagation



CS231n course from Stanford University

Backpropagation using Local Gradients and Chain Rule

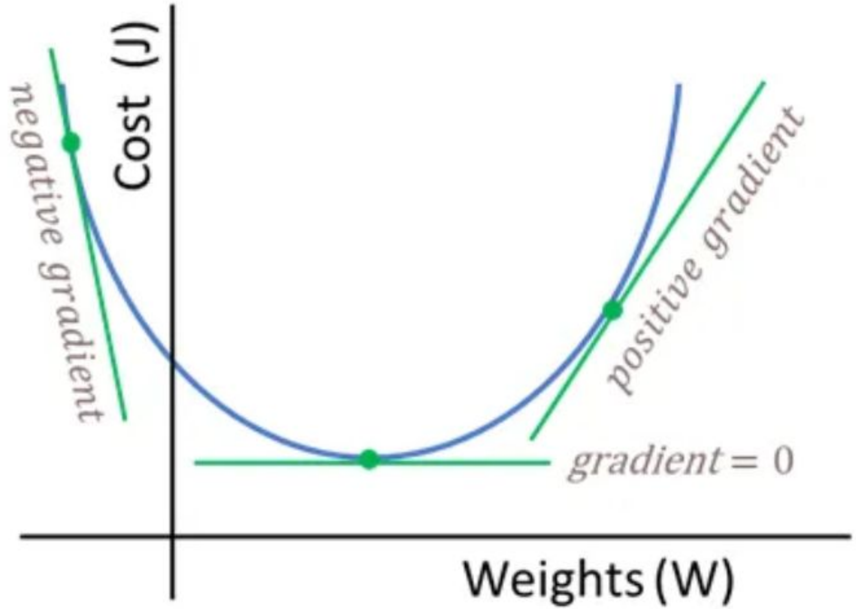




- ➢ The Chain Rule is implemented with the help of **local gradients**.
- ➢ We **recursively multiply** the local derivatives.
- ➢ Backpropagation is a **recursive** application of the chain rule backwards through the **computation graph**.

CS231n course from Stanford University

Cost function



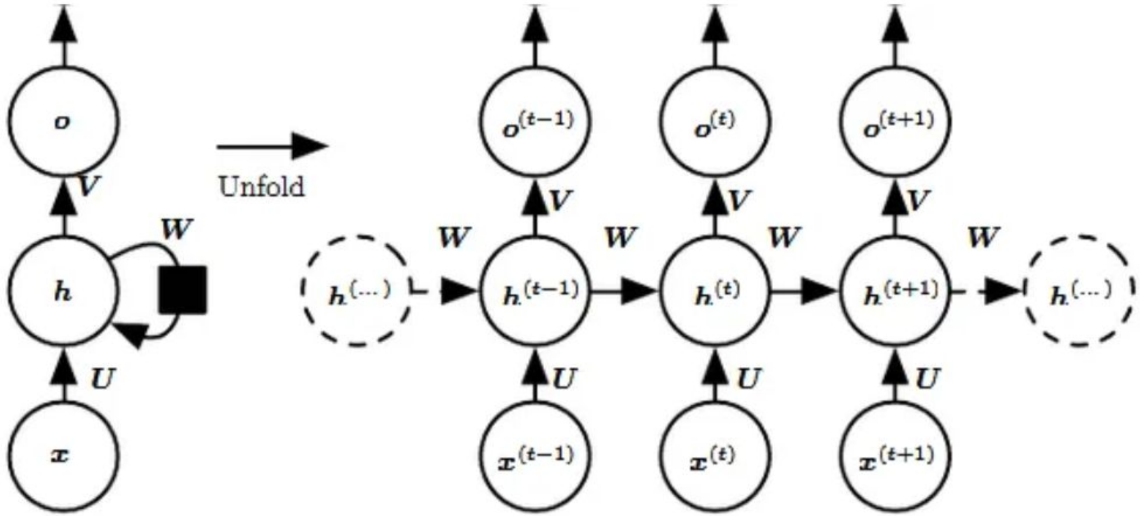
Weights update

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

➢ **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**.

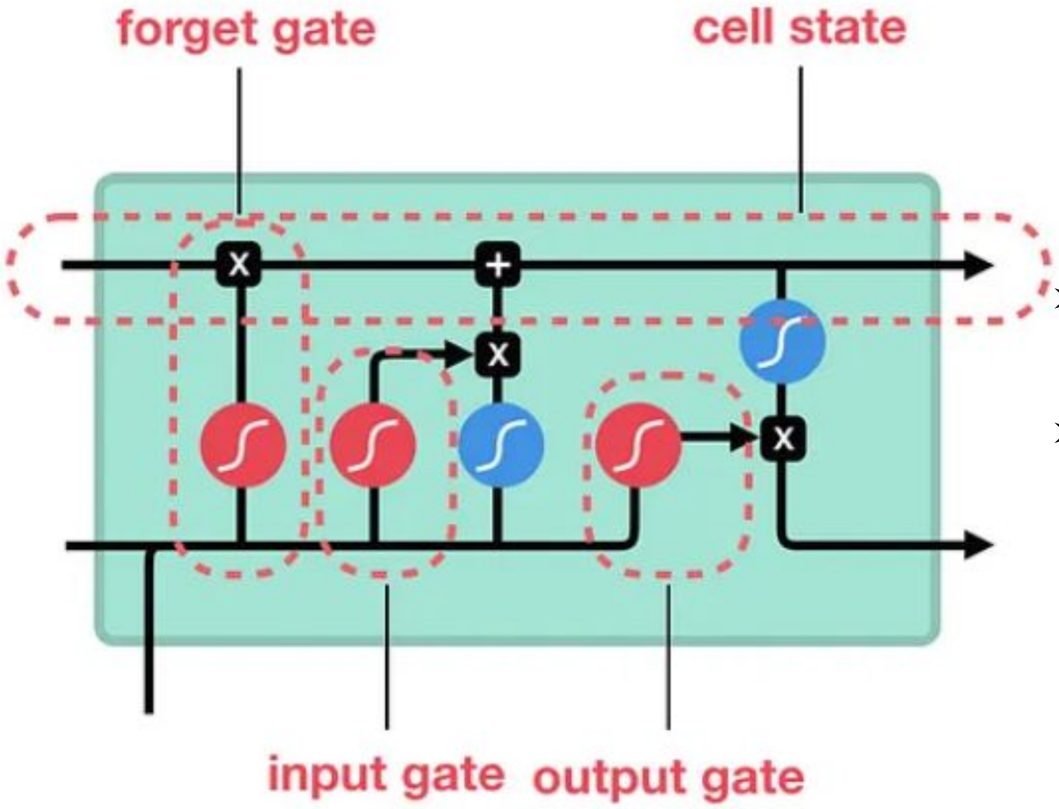
➢ The **weights are updated** in the direction of the **negative** gradient, so that the **cost function is minimized** as much as possible.

Sequential nature of Recurrent Neural Networks (RNNs)



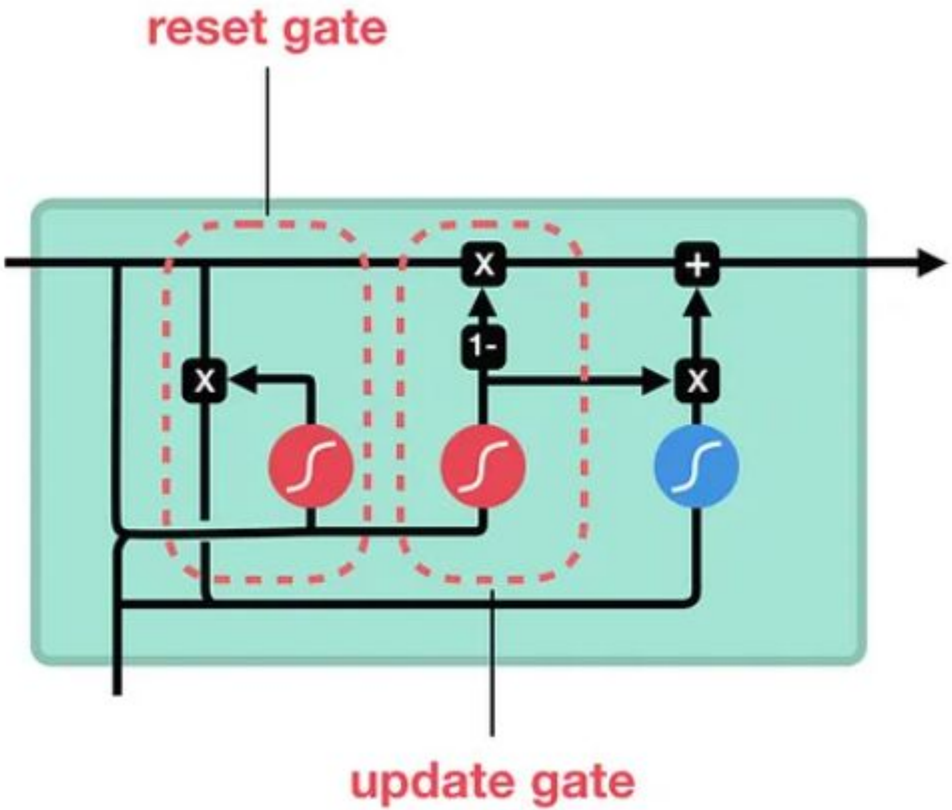
- ➢ By **unfolding** the feedback loop in time, we become aware of the complexity of these networks. It is as if we train a **very deep network** and that is why they are harder to train.
- ➢ With **RNNs** things are done **sequentially** => **deep** graph structure.
- ➢ With **Transformers** things happen **parallely** => **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)



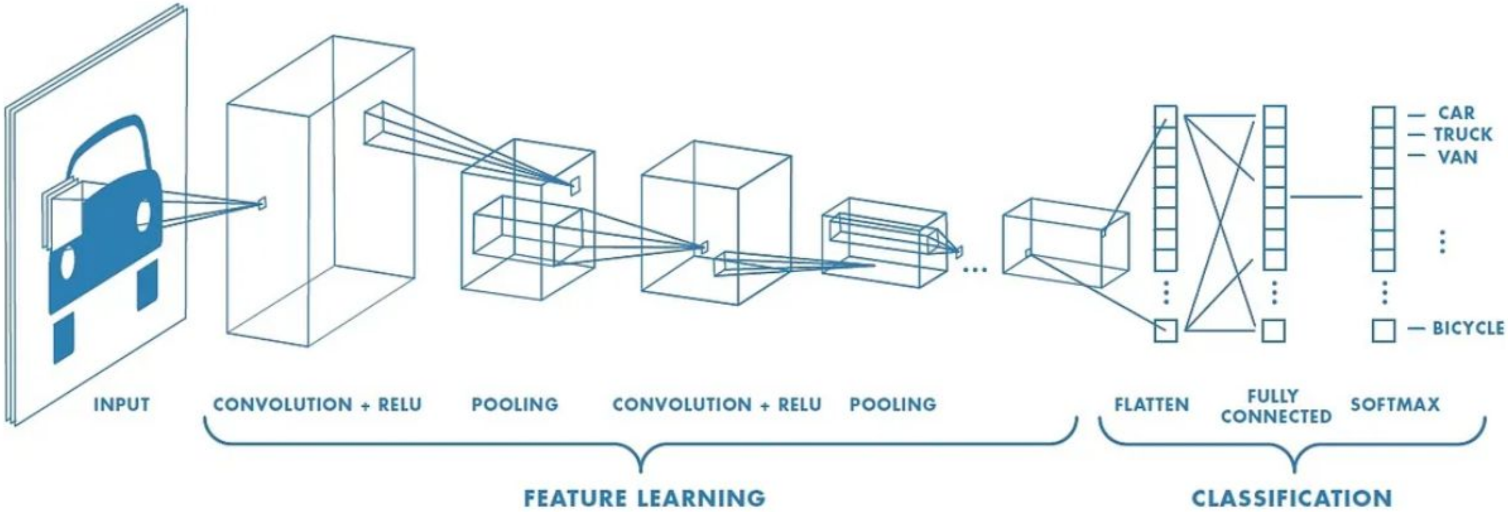
- ➢ Contains special **gates** that address the problem of **vanishing gradients**.
- ➢ Addresses the problem of **exploding gradients**.

Gated Recurrent Units (GRUs)



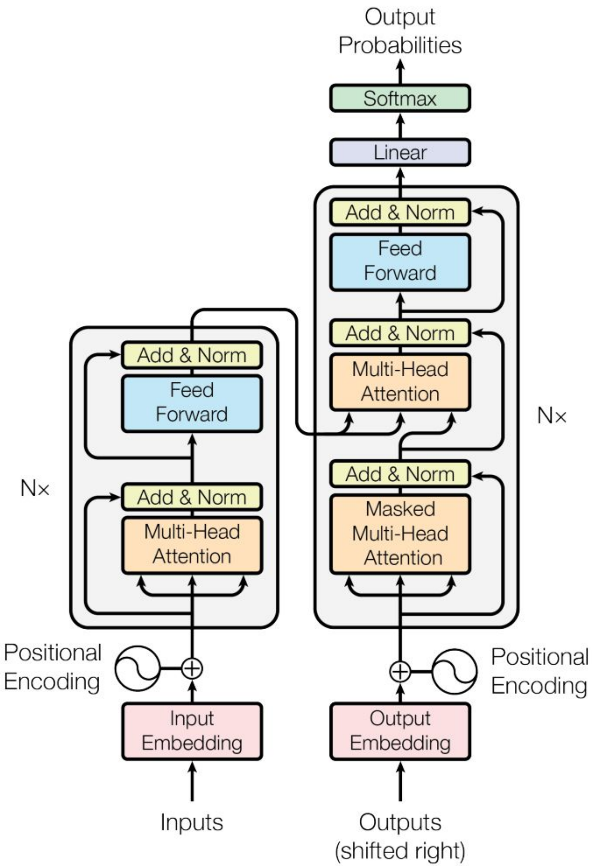
- ➢ Generally considered **faster** than LSTMs.
- ➢ In practice => **similar outcomes to** what **LSTM** provides.

Convolutional Neural Networks (CNNs)



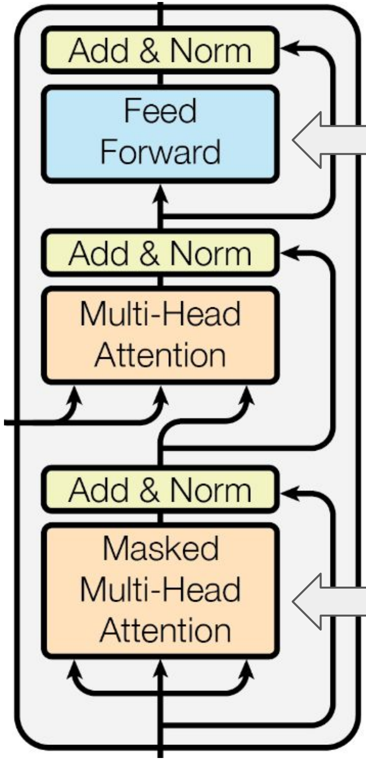
- Generally applied in **computer vision** tasks, i.e. **2D image focused, not time sequence data**.
- We can use **3D CNNs to handle sequences of data**, where 3rd dimension is time. Here we talk about a cube kernel, instead of a plane 2D kernel.
- CNNs are very amenable to **parallelism**.

Transformers



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

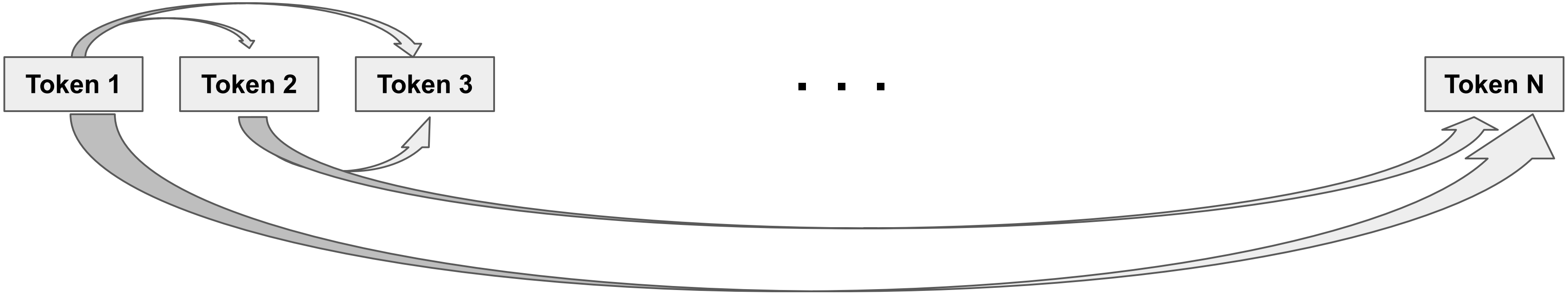
Transformer Block



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**).

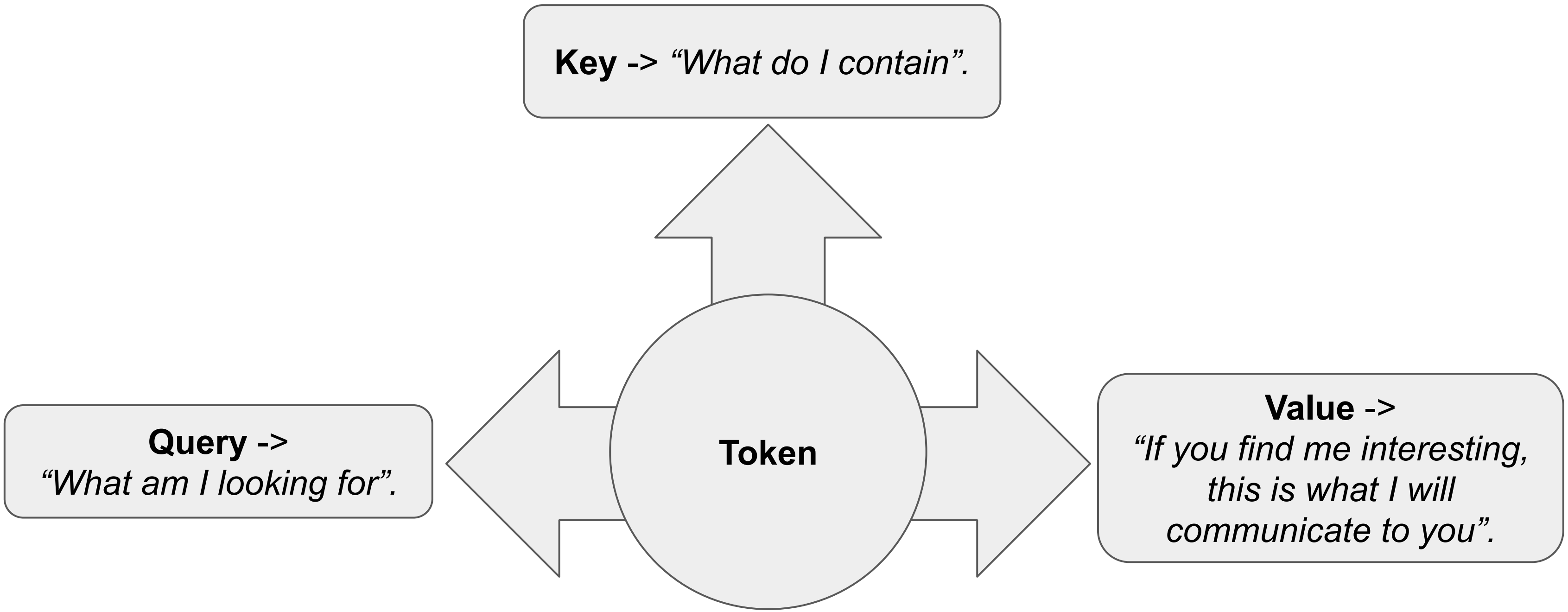
Self-Attention



➢ **All tokens communicate with one another**.

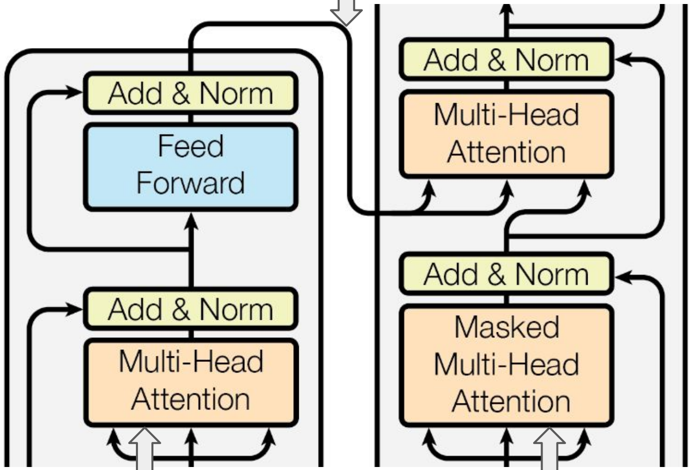
➢ This is computationally expensive because **each token has to look at every other token** to compute an **attention score / attention weight**.

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.

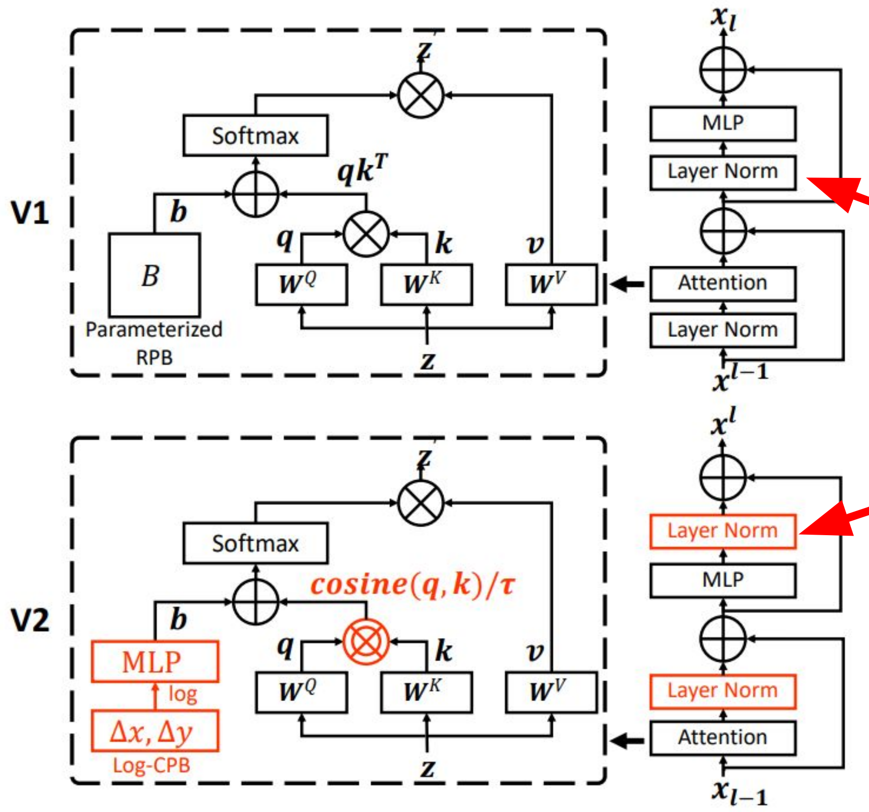
Mathematically Expressing Self-Attention

- ➢ The dot product **Query ▪ Key** is the attention score.

- ➢ 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

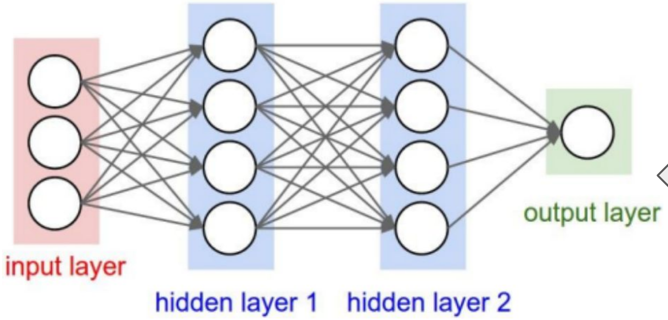


Notice the **pre**-Layer Norm
vs
post-Layer Norm
switch.

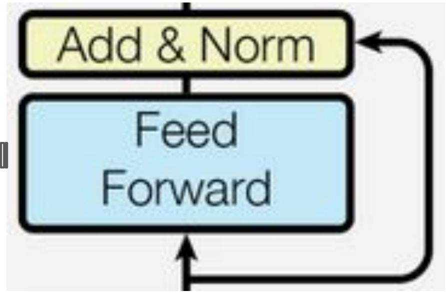
"Swin Transformer V2: Scaling Up Capacity and Resolution" by Liu et al.

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 (BOG)**.
- ➢ 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 simply **added literally** 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 ends up performing. Check out the Swin Transformer paper for empirical proof.*

ChatGPT Pipeline

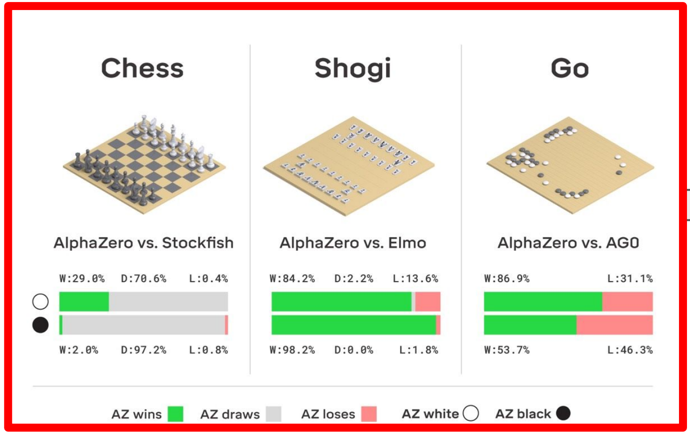
1) Pretraining the **base model**.

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.



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

Pretraining the Foundational Model

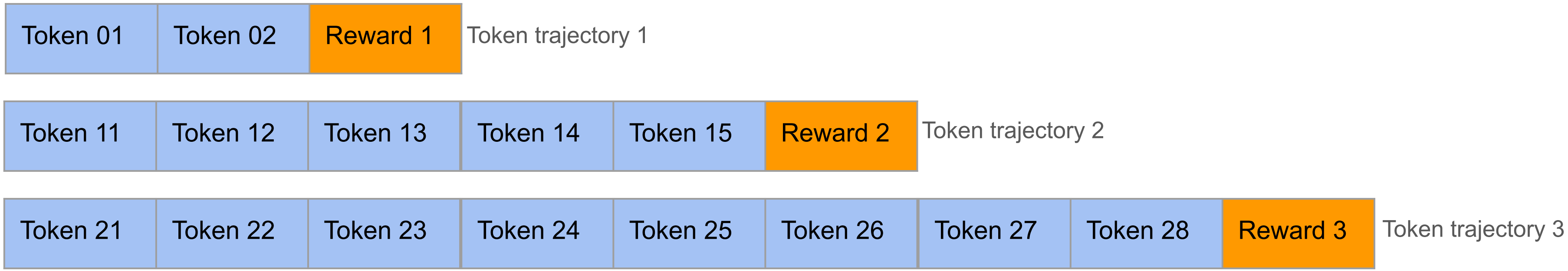
- ➢ Use **raw data** to train a **Base Model**.
- ➢ The dataset is **huge** => potentially low quality, but **very large quantity**.
- ➢ We obtain a **document completer** in the end.
- ➢ Thousands of GPUs work in parallel (ex: 1000-2000 A100 GPUs)

Supervised Fine Tuning (SFT stage)

- **Low quantity, 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 (ex: 1-100).
- The outcome is the so-called **SFT model**.

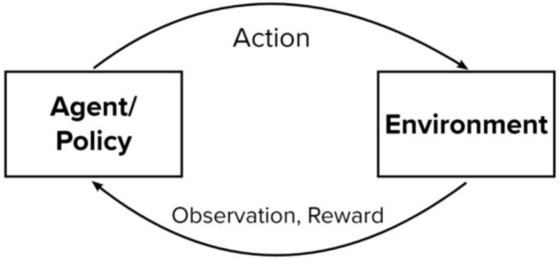
Reward Modeling

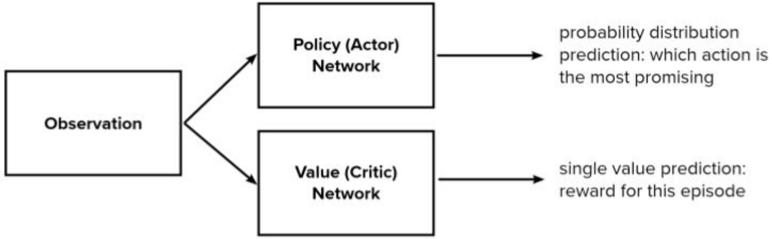
- ➢ Ask the SFT model to produce **multiple answers** per prompt.
- ➢ Ask contractors to carefully **rank these answers**.
- ➢ Train a reward model on these rankings.
- ➢ Order of 1 to 100 GPUs for training.
- ➢ The outcome is the so-called **Reward Model / RM** => **Evaluates token trajectories.**



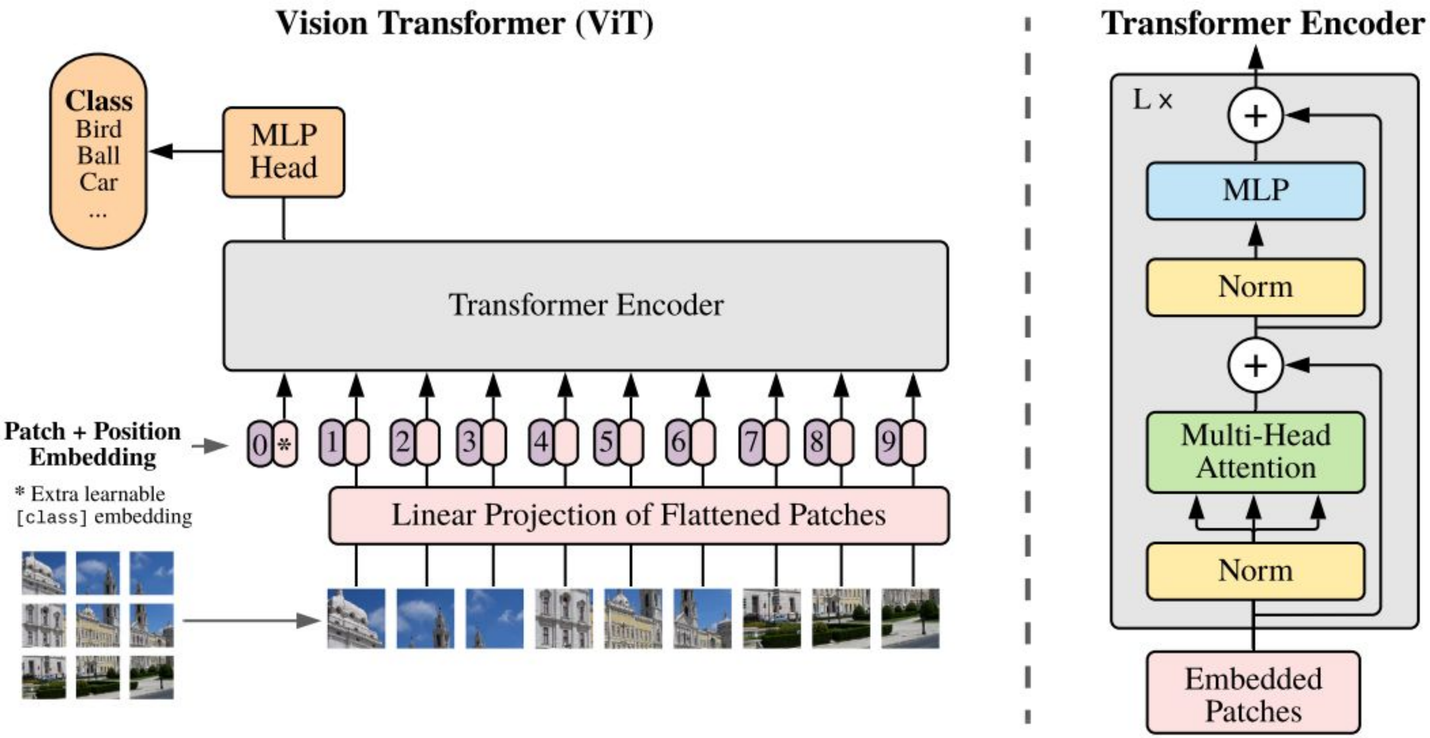
Reinforcement Learning

- ➢ Train a **PPO** algorithm (**Proximal Policy Optimization**).
- ➢ Use the previously trained reward model to **evaluate the reward**.
- ➢ PPO will have the job 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**).



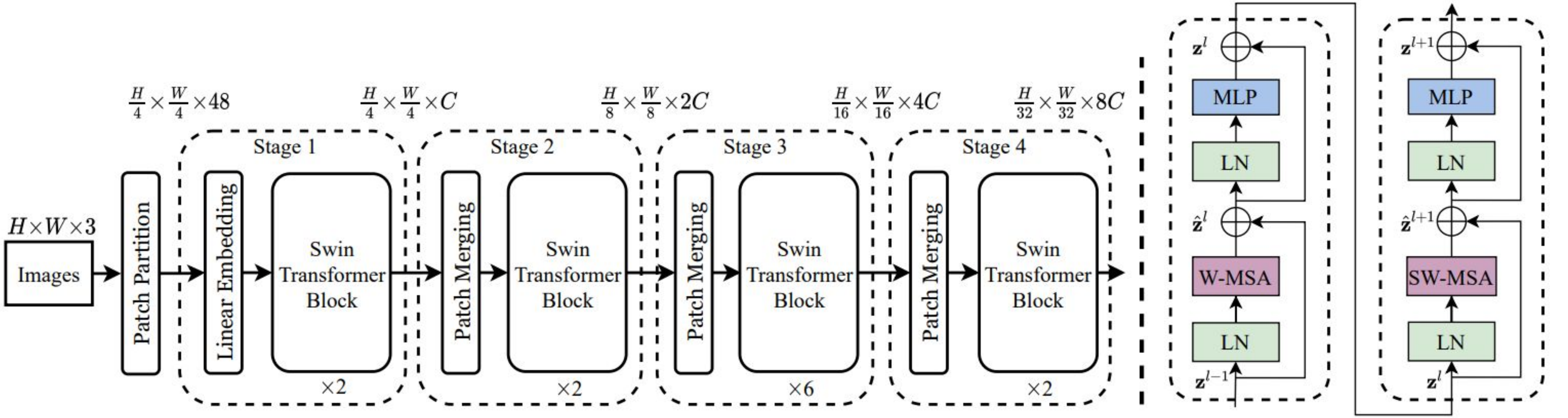


Vision Transformer / ViT



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

Swin Transformer



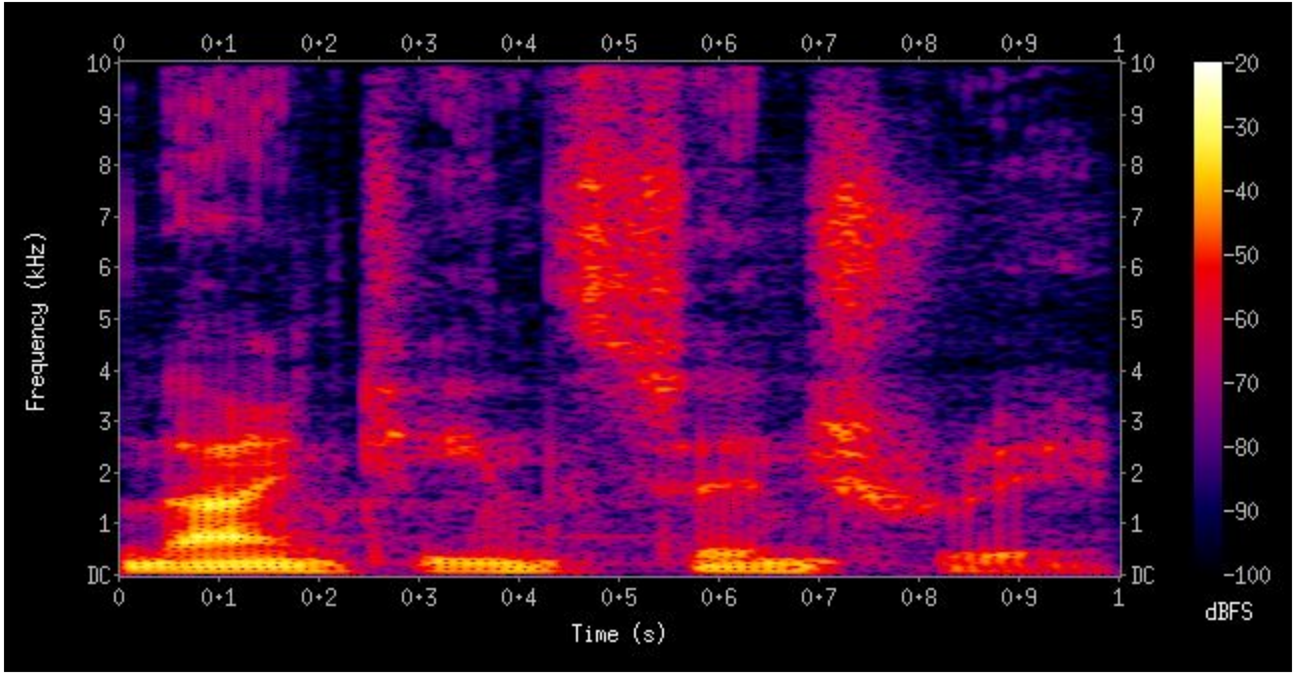
(a) Architecture

(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 addressed 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

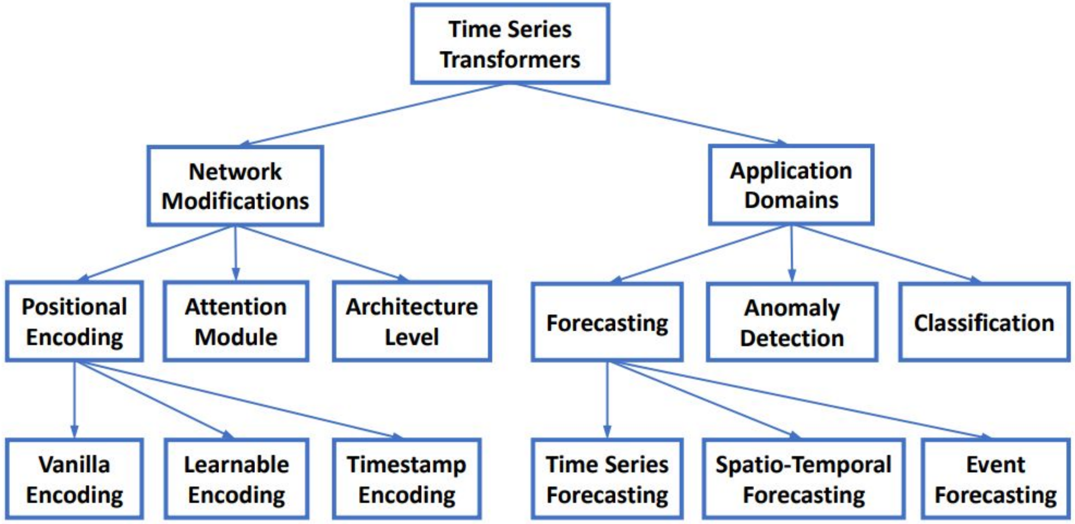


Figure 1: Taxonomy of Transformers for time series modeling from the perspectives of network modifications and application domains.

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

Mini-GPT Qualitative Results

Trained on DKRZ server:

```
WARWICK:
Mine eyes have bands the cause of block.

RICHARD:
Up Montague, and in the opin stroke,
Her hath hand with the sharp of Edward's care.

RICHARD:
But repetitor of thy brothers are postless of fight
Thy in habit wars, for presumpting Bishop's wrongs.

WARWICK:
And live peace, thou know'st what of Henry William come?

SOMERSET:
The which such debt sworns, giving loood of loving men:
Ere ye he suspicious, turns the devour death?
And spurn my kning bead of love to his friends,
I'ld up their witten in a wolvesse slay,
Standing from that in chiefes the befalls!

Second Murderer:
The talts, and did stand and call the surping lass;
More weightily in a jarent air,
More than my years, and being toilous,
Making weep and from mine head annest,
Courage of their rotten foot and tithe,
You cannot speak love the frown of y hands
The stread on my terrich! Valia, my dam:
More passion clergine of those first black ears:
Come, bring puling impositials that which
Rome or Rather's busine bow: this
```

Also, checkout -> https://www.nytimes.com/interactive/2023/04/26/upshot/gpt-from-scratch.html

Training on GPU

```
Every 2.0s: nvidia-smi

Tue Jul 11 10:20:59 2023
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 535.54.03              Driver Version: 535.54.03    CUDA Version: 12.2     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  NVIDIA A100-SXM4-40GB          Off | 00000000:03:00.0 Off |                    0 |
| N/A   69C    P0             244W / 400W |   4355MiB / 40960MiB |     98%      Default |
|                                         |                      |             Disabled |
+-----------------------------------------+----------------------+----------------------+
|   1  NVIDIA A100-SXM4-40GB          Off | 00000000:44:00.0 Off |                    0 |
| N/A   49C    P0              63W / 400W |      8MiB / 40960MiB |      0%      Default |
|                                         |                      |             Disabled |
+-----------------------------------------+----------------------+----------------------+
|   2  NVIDIA A100-SXM4-40GB          Off | 00000000:84:00.0 Off |                    0 |
| N/A   48C    P0              59W / 400W |      8MiB / 40960MiB |      0%      Default |
|                                         |                      |             Disabled |
+-----------------------------------------+----------------------+----------------------+
|   3  NVIDIA A100-SXM4-40GB          Off | 00000000:C4:00.0 Off |                    0 |
| N/A   48C    P0              59W / 400W |      8MiB / 40960MiB |      0%      Default |
|                                         |                      |             Disabled |
+-----------------------------------------+----------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
|    0   N/A  N/A    126312      G   /usr/libexec/Xorg                            23MiB |
|    0   N/A  N/A   1883655      C   python                                     4308MiB |
+---------------------------------------------------------------------------------------+
```

Transformer Hyperparameters

```python
# hyperparameters
batch_size = 64 # how many independent sequences will we process in parallel?
block_size = 256 # what is the maximum context length for predictions?
max_iters = 5000
eval_interval = 500
learning_rate = 3e-4
device = 'cuda' if torch.cuda.is_available() else 'cpu'
eval_iters = 200
n_embd = 384
n_head = 6
n_layer = 6
dropout = 0.2
```

Train Info Logs

```
10.788929 M parameters
step 0: train loss 4.2221, val loss 4.2306
step 500: train loss 1.7550, val loss 1.9111
step 1000: train loss 1.3907, val loss 1.6016
step 1500: train loss 1.2679, val loss 1.5275
step 2000: train loss 1.1857, val loss 1.4956
step 2500: train loss 1.1227, val loss 1.4960
step 3000: train loss 1.0720, val loss 1.4844
step 3500: train loss 1.0207, val loss 1.4968
step 4000: train loss 0.9595, val loss 1.5057
step 4500: train loss 0.9102, val loss 1.5299
step 4999: train loss 0.8607, val loss 1.5576
```

Starts to overfit

Takeaways

- ➢ 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**.
- ➢ **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
- ➢ Slides 13, 14, 17, 20, 21 Transformer components were taken from “Attention is all you need” paper, by Vaswani et al.
- ➢ Slide 19: “Swin Transformer V2: Scaling Up Capacity and Resolution” paper, by Liu et al.
- ➢ Slide 22: https://arstechnica.com/science/2018/12/move-over-alphago-alphazero-taught-itself-to-play-three-different-games/
- ➢ Slide 26: https://odsc.com/blog/reinforcement-learning-with-ppo/
- ➢ Slide 27: “An image is worth 16x16 words: Transformers for image recognition at scale” paper, by Dosovitskiy et al.
- ➢ Slide 28: “Swin Transformer: Hierarchical Vision Transformer using Shifted Windows” paper, by Liu et al.
- ➢ Slide 29: https://en.wikipedia.org/wiki/Short-time_Fourier_transform#/media/File:Spectrogram-19thC.png
- ➢ Slide 30: “Transformers in Time Series: A Survey” paper, by Wen et al.

Appendix: 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]])
```

Appendix: Self-Attention Snippet Version 2

```
# 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
```

Appendix: 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
```