Attention

deep learning 5

R. Grouls, 20 mei 2022

Metrics

Confusion matrix

There are two types of error:

- You predict positive, but it is actual false
- You predict negative, but it is actual true

	actual positive	actual negative
predicted positive	True Positive	False Positive
predicted negative	False Negative	True Negative

Precision vs Recall

$$recall = \frac{TP}{TP + FN}$$

$$precision = \frac{TP}{TP + FP}$$

	actual positive	actual negative
predicted positive	True Positive	False Positive
predicted negative	False Negative	True Negative

Precision vs Recall

You are having a kids party where you go swimming.

At the end, you need to gather everyone

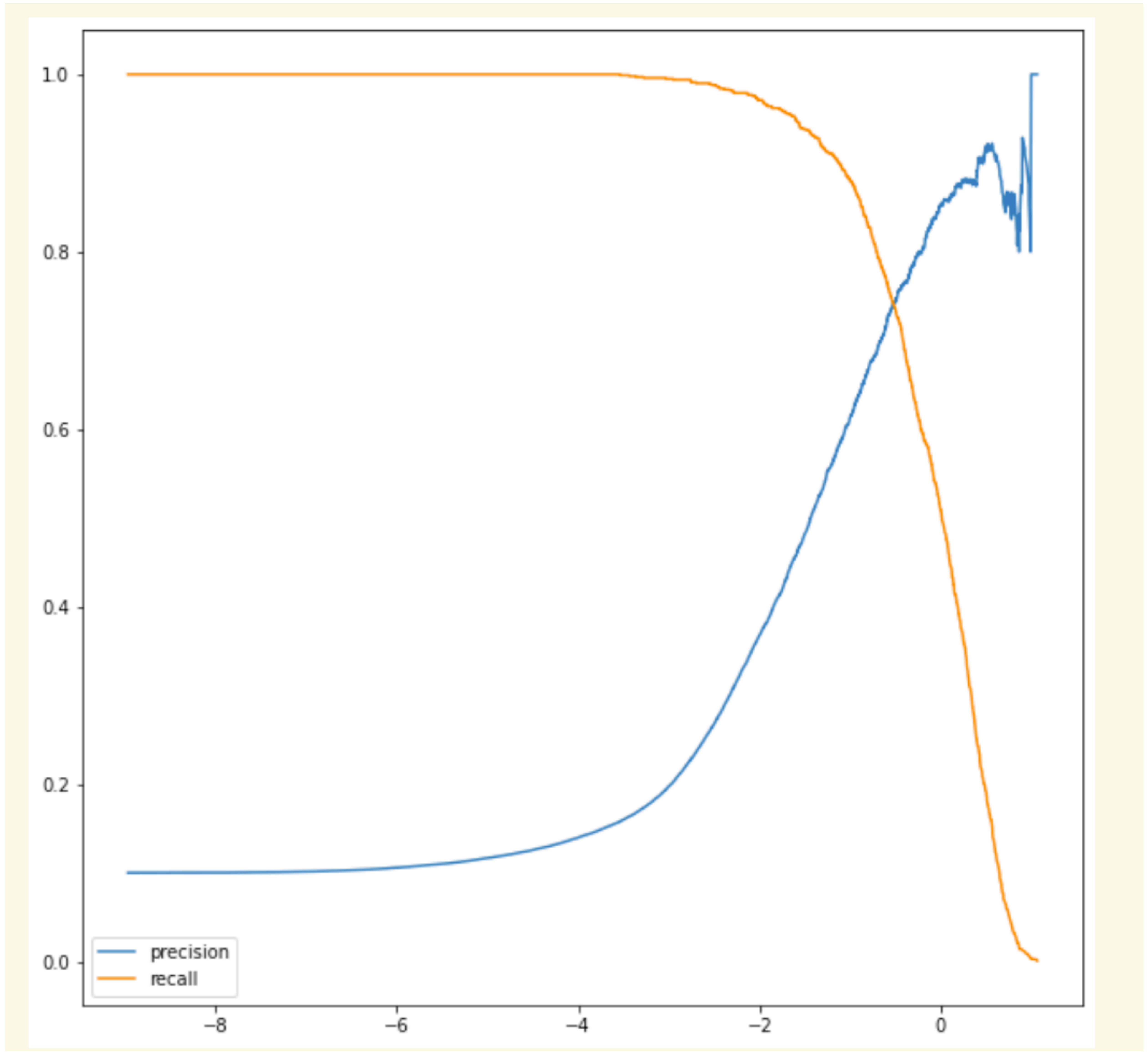
1. You leave one child behind, that should have come along
2. You add an additional child, that isn’t from the party

Which of these is a precision error? And which is a recall error?i

Trade off

Precision vs Recall is always a trade off.

Why is that?

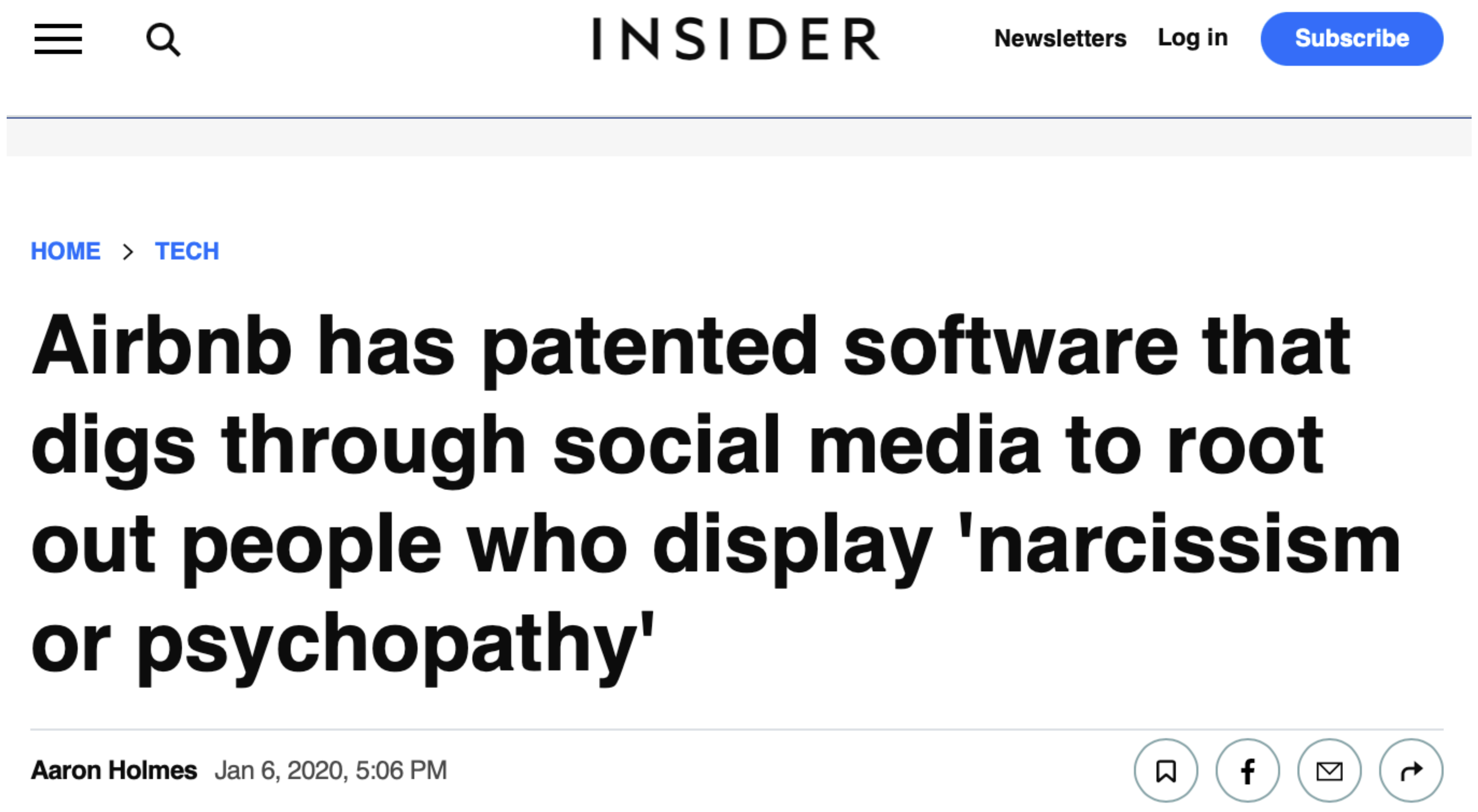


Confusion matrix ethics

- What are precision vs recall errors with the Airbnb algorithm?

Which mix is best for:

- Guests renting a place
- Hosts offering a place
- Airbnb revenue



Airbnb has patented software that digs through social media to root out people who display 'narcissism or psychopathy'

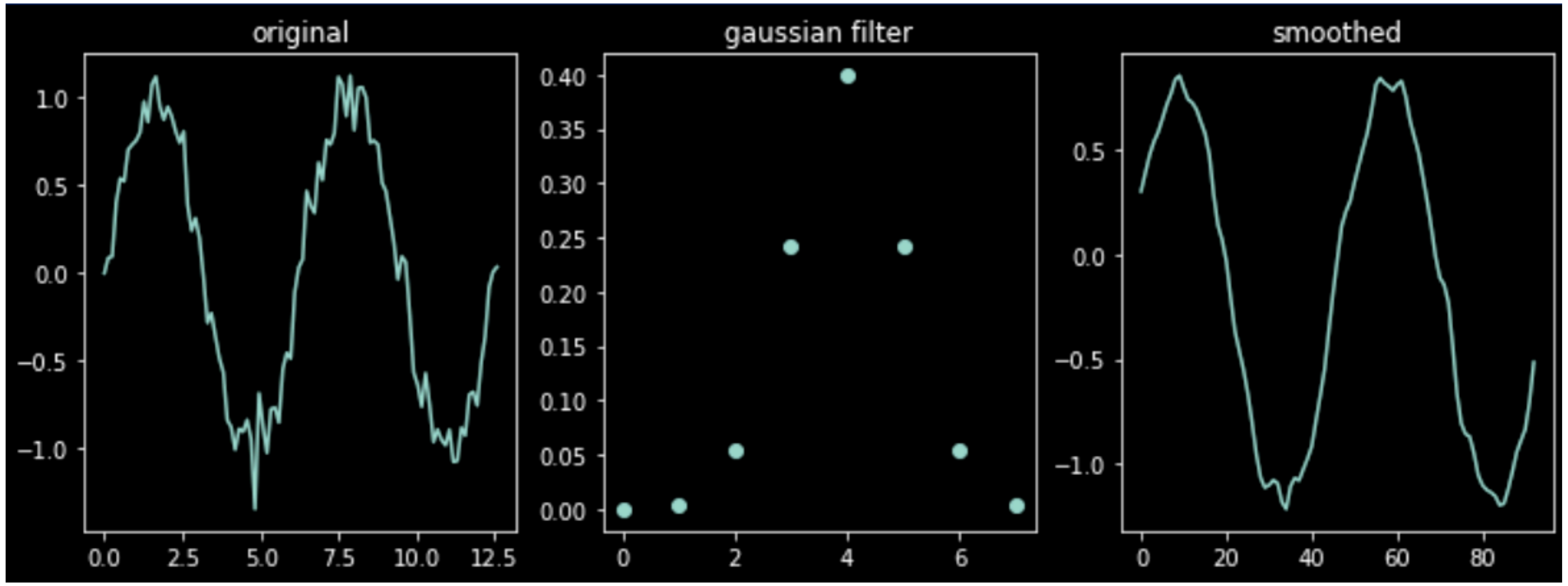
Aaron Holmes Jan 6, 2020, 5:06 PM

Attention

Motivation

local weighing is not enough

- We have been using convolutions
- These look at local context
- But what about more distant relations?



Attention is all you need

- “Attention is all you need” was published in 2017 https://arxiv.org/abs/1706.03762
- This architecture replaced the RNN, and revolutionized the NLP models. It is part of most new groundbreaking algorithms

Mixing in context

Intuition

Compare

The bank of the river

The cashier at the bank

We can differentiate between the different meanings of the word *bank* because we somehow “mix in” the meaning of the rest of the sentence.

We would want to add some of the meaning of “river” and “cashier” to the meaning of “bank”

You could also say, we are paying *attention* to other words to give *bank* it’s meaning

Mixing in context

Mathematical

Let ξ_i be the vector that describes the semantic meaning of word i.

We can imagine the we want to “mix in” other words, some more than others. A simple way would be with multiplication.

This way, we obtain a vector of numbers, one for every word in the sequence.

$$\begin{array}{cccc} & \xi_1 & \xi_2 & \xi_3 \\ \xi_1 & \xi_1\xi_1 & \xi_1\xi_2 & \xi_1\xi_3 \end{array}$$

Reweighing relevance

For the meaning of ξ_1, let ξ_2 be irrelevant, and ξ_3 be highly relevant.

We would want $\xi_1\xi_2$ to be closer to zero, and $\xi_1\xi_3$ to be closer to one.

If we normalize all weights with a softmax, we make sure everything adds to 1

We can now reweigh the original ξ_1 to obtain $\tilde{\xi}_1$

And it will contain a little bit of ξ_2 context but much more of ξ_3 context

$$W = softmax(\begin{bmatrix}\xi_1\xi_1 & \xi_1\xi_2 & \xi_1\xi_3\end{bmatrix})$$

$$W = [0.4 \quad 0.1 \quad 0.4]$$

$$\tilde{\xi}_1 = w_1\xi_1 + w_2\xi_2 + w_3\xi_3$$

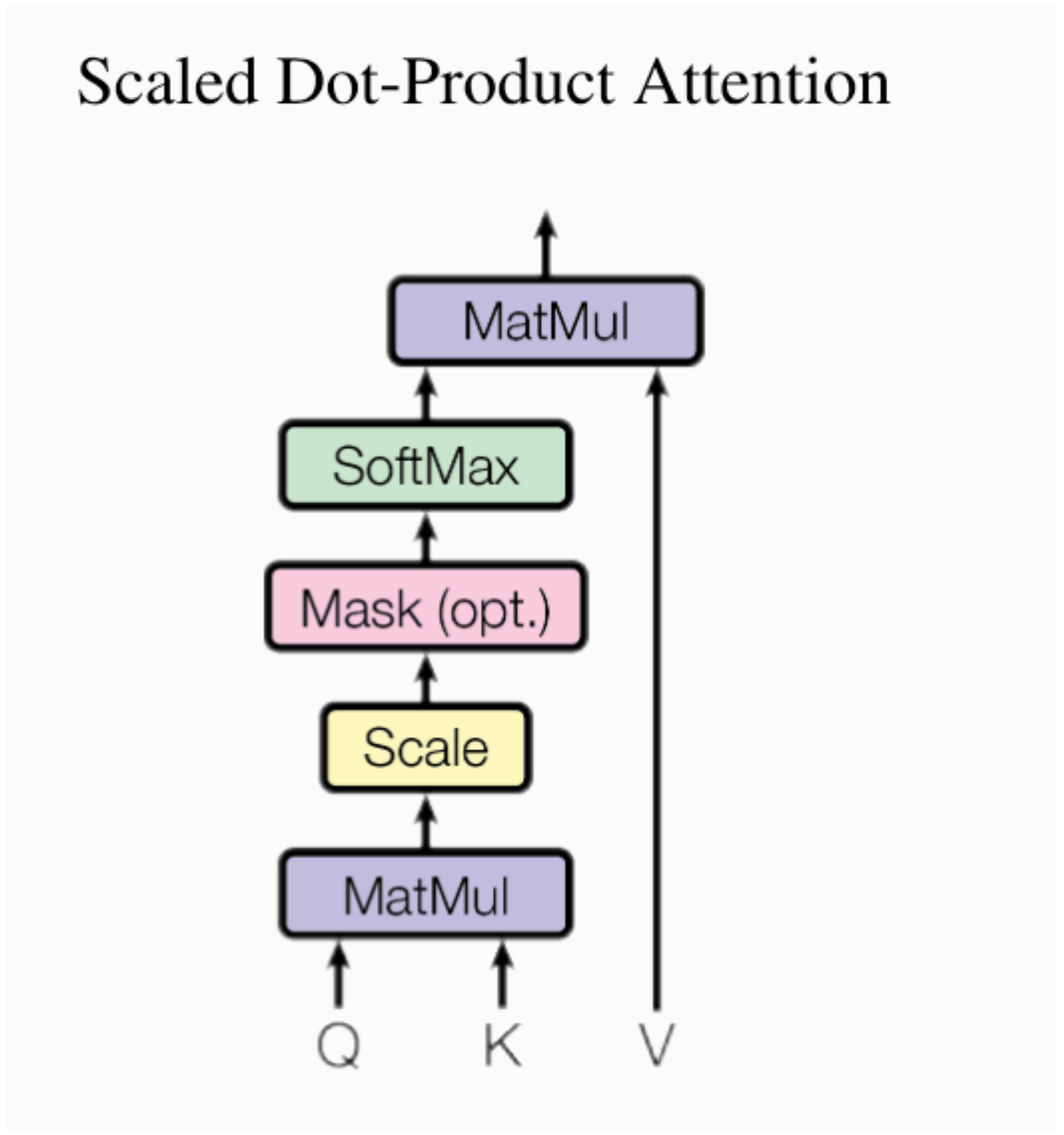
Attention

We can do this for complete matrix at once. We take two copies of the embeddings with dimensions *(sequence, dimensions)*, the matrices Q and K.

$$weights = softmax(QK^T)$$

We obtain a weights matrix with dimensions *(sequence, sequence)*, which we now want to multiply with the original embeddings V for the attention reweighing.

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

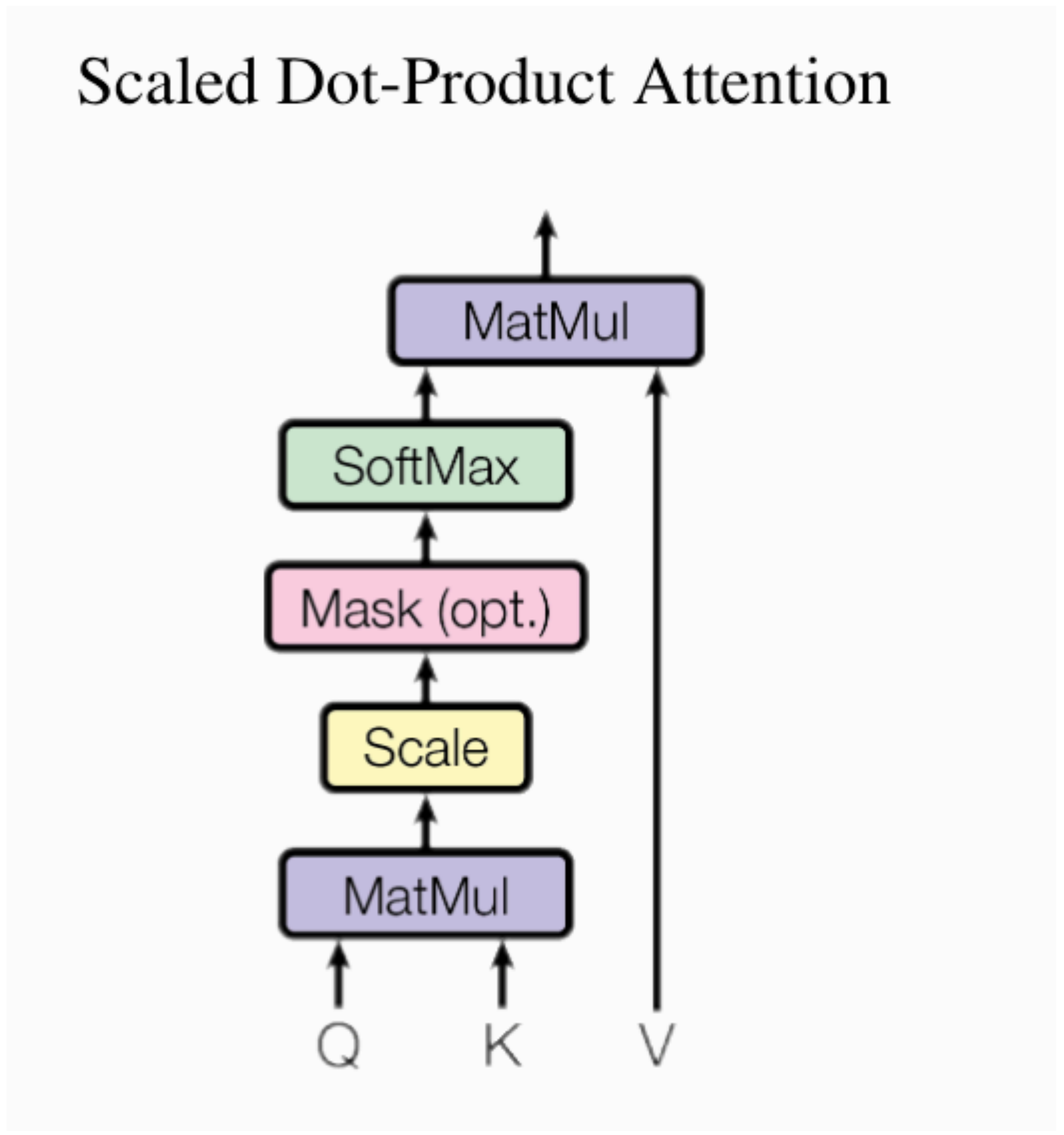


Scaling and masking

The scaling factor $\frac{1}{\sqrt{d_k}}$ is crucial to keep an appropriate variance; otherwise, the softmax wil saturate to 1 for one arbitrary element.

The optional mask (see image) can be used to block out padding values, or causal values in the case of timeseries.

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



Attention preserves dimensions

Input and output typically the same dimensions

Let Q, K, V be Query, Key and Value matrices. Let s be the length of the sequence, and d the dimensionality. We denote the dimensions of the matrix with $\mathbb{R}^{s \times d}$

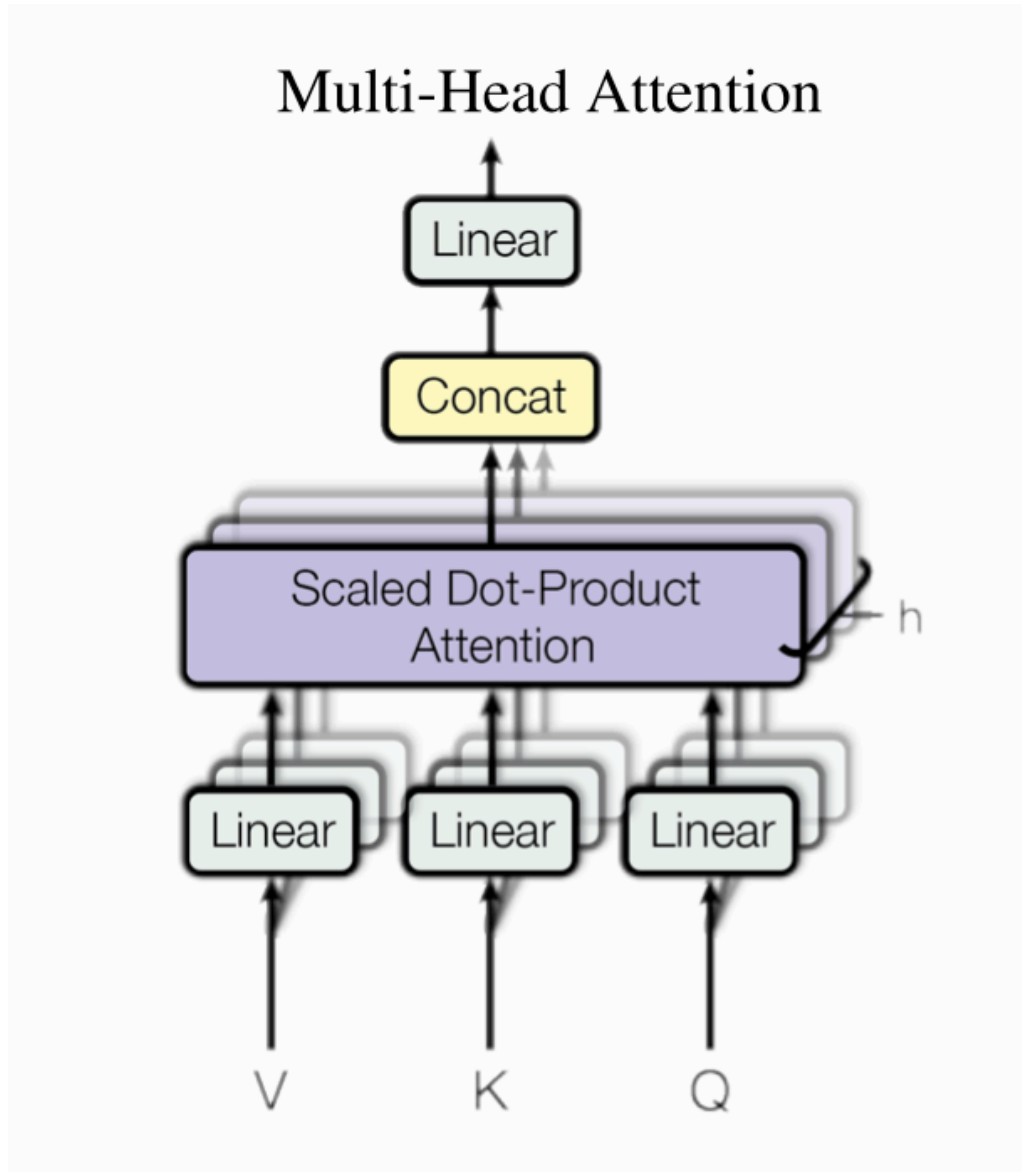
with $Q, K, V \in \mathbb{R}^{s \times d}$ we get $QK^T \in \mathbb{R}^{s \times s}$

This is because the matrix multiplication QK^T with dimensions (s, d) and (d, s) gives us dimensions (s, s)

In the second matrix multiplication $\left(QK^T\right) V$ we multiply (s, s) and (s, d) so we end up with $\mathbb{R}^{s \times d}$ again.

Multi-Head

- Multi-Head Attention is simply splitting up the embedding into smaller pieces, doing attention on every piece, and concatenating everything at the end.



Feed Forward

The model includes a feed forward network.

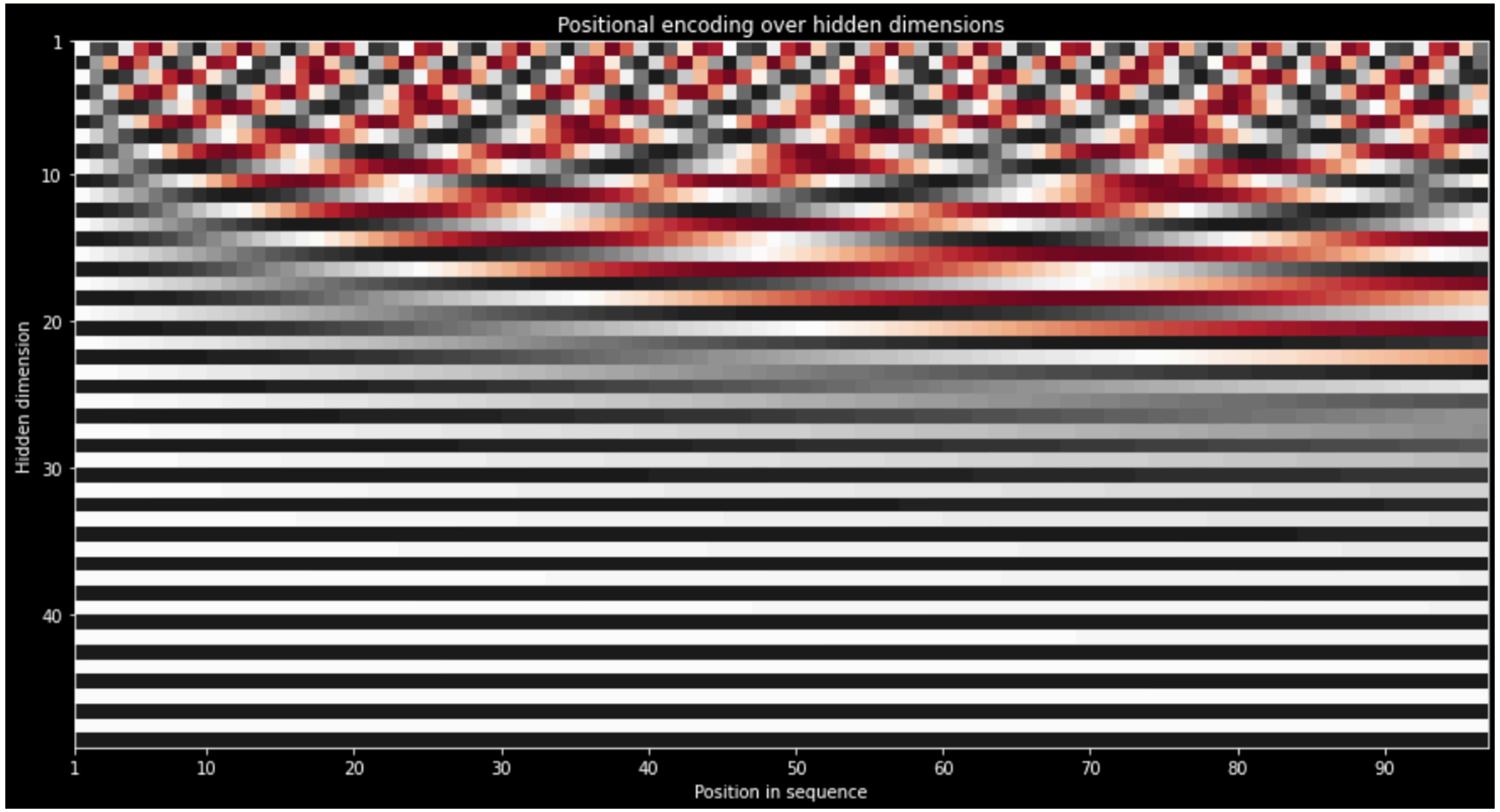
$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2$$

This is just another way of writing down what we have been working with since lesson 1:

- a two layer linear network $f(x) = Wx + b$
- with a $relu(x) = max(0, x)$ after the first linear layer.

Positional Encoding

- A downside is that the model will have no idea about position in time.
- PE solves this by adding sinewaves with different frequencies together
- Note that this is exactly how our time-system works (sec, min, hour, etc)
- This is enough for the model to figure out relative positions in time.



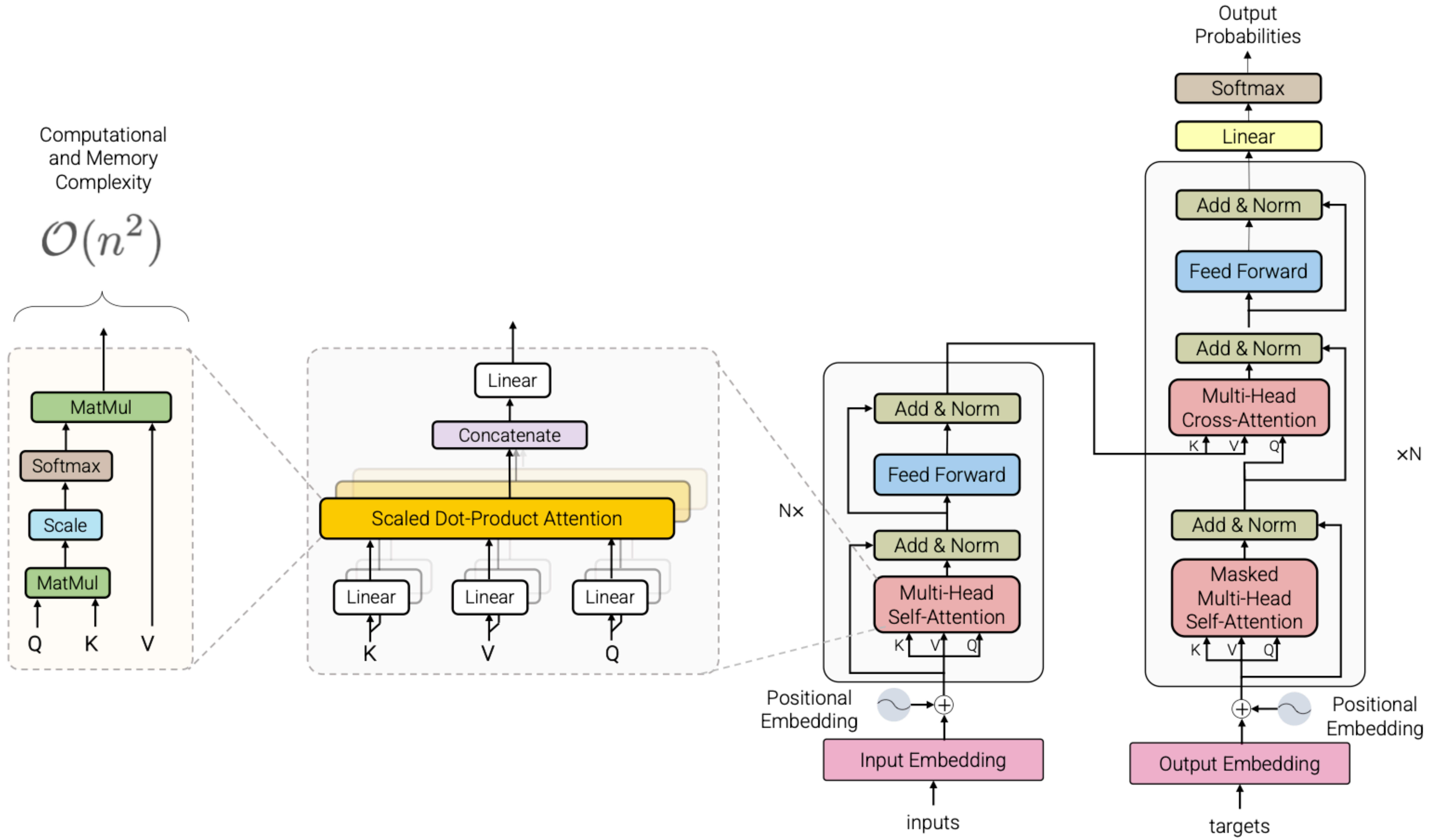
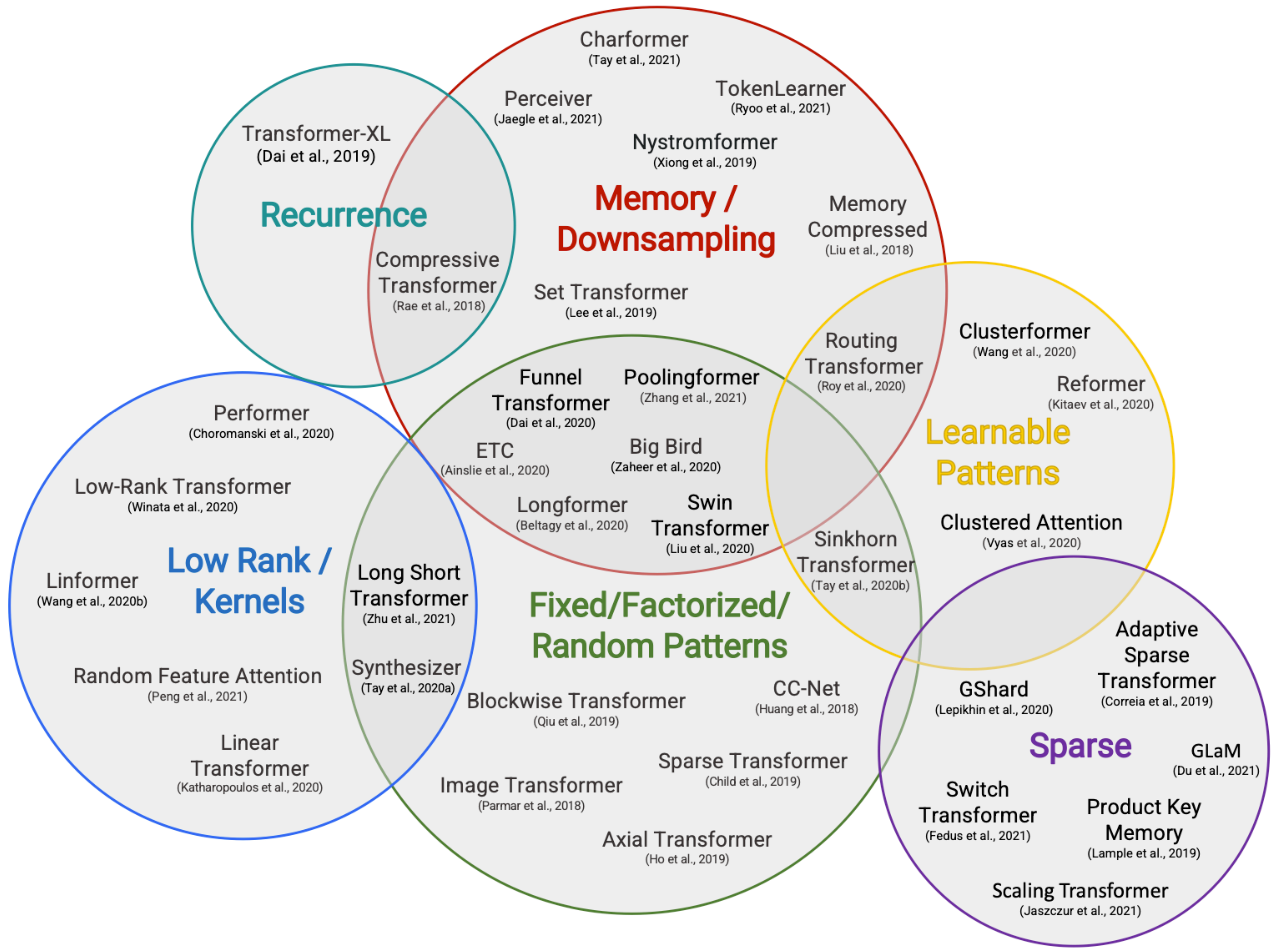


Figure 1: Architecture of the standard Transformer (Vaswani et al., 2017)

Efficient transformers

Model / Paper	Complexity	Decode	Class
Memory Compressed (Liu et al., 2018)	$\mathcal{O}(N_c^2)$	✓	FP+M
Image Transformer (Parmar et al., 2018)	$\mathcal{O}(N.m)$	✓	FP
Set Transformer (Lee et al., 2019)	$\mathcal{O}(kN)$	✗	M
Transformer-XL (Dai et al., 2019)	$\mathcal{O}(N^2)$	✓	RC
Sparse Transformer (Child et al., 2019)	$\mathcal{O}(N\sqrt{N})$	✓	FP
Reformer (Kitaev et al., 2020)	$\mathcal{O}(N \log N)$	✓	LP
Routing Transformer (Roy et al., 2020)	$\mathcal{O}(N\sqrt{N})$	✓	LP
Axial Transformer (Ho et al., 2019)	$\mathcal{O}(N\sqrt{N})$	✓	FP
Compressive Transformer (Rae et al., 2020)	$\mathcal{O}(N^2)$	✓	RC
Sinkhorn Transformer (Tay et al., 2020b)	$\mathcal{O}(B^2)$	✓	LP
Longformer (Beltagy et al., 2020)	$\mathcal{O}(n(k+m))$	✓	FP+M
ETC (Ainslie et al., 2020)	$\mathcal{O}(N_g^2 + NN_g)$	✗	FP+M
Synthesizer (Tay et al., 2020a)	$\mathcal{O}(N^2)$	✓	LR+LP
Performer (Choromanski et al., 2020a)	$\mathcal{O}(N)$	✓	KR
Funnel Transformer (Dai et al., 2020)	$\mathcal{O}(N^2)$	✓	FP+DS
Linformer (Wang et al., 2020c)	$\mathcal{O}(N)$	✗	LR
Linear Transformers (Katharopoulos et al., 2020)	$\mathcal{O}(N)$	✓	KR
Big Bird (Zaheer et al., 2020)	$\mathcal{O}(N)$	✗	FP+M
Random Feature Attention (Peng et al., 2021)	$\mathcal{O}(N)$	✓	KR
Long Short Transformers (Zhu et al., 2021)	$\mathcal{O}(kN)$	✓	FP + LR
Poolingformer (Zhang et al., 2021)	$\mathcal{O}(N)$	✗	FP+M
Nyströmformer (Xiong et al., 2021b)	$\mathcal{O}(kN)$	✗	M+DS
Perceiver (Jaegle et al., 2021)	$\mathcal{O}(kN)$	✓	M+DS
Clusterformer (Wang et al., 2020b)	$\mathcal{O}(N \log N)$	✗	LP
Luna (Ma et al., 2021)	$\mathcal{O}(kN)$	✓	M
TokenLearner (Ryoo et al., 2021)	$\mathcal{O}(k^2)$	✗	DS
Adaptive Sparse Transformer (Correia et al., 2019)	$\mathcal{O}(N^2)$	✓	Sparse
Product Key Memory (Lample et al., 2019)	$\mathcal{O}(N^2)$	✓	Sparse
Switch Transformer (Fedus et al., 2021)	$\mathcal{O}(N^2)$	✓	Sparse
ST-MoE (Zoph et al., 2022)	$\mathcal{O}(N^2)$	✓	Sparse
GShard (Lepikhin et al., 2020)	$\mathcal{O}(N^2)$	✓	Sparse
Scaling Transformers (Jaszczur et al., 2021)	$\mathcal{O}(N^2)$	✓	Sparse
GLaM (Du et al., 2021)	$\mathcal{O}(N^2)$	✓	Sparse



Charformer (Tay et al., 2021)
Perceiver (Jaegle et al., 2021)
TokenLearner (Ryoo et al., 2021)
Transformer-XL (Dai et al., 2019)
Nystromformer (Xiong et al., 2019)
Recurrence
Memory / Downsampling
Memory Compressed (Liu et al., 2018)
Compressive Transformer (Rae et al., 2018)
Set Transformer (Lee et al., 2019)
Routing Transformer (Roy et al., 2020)
Clusterformer (Wang et al., 2020)
Funnel Transformer (Dai et al., 2020)
Poolingformer (Zhang et al., 2021)
Reformer (Kitaev et al., 2020)
Performer (Choromanski et al., 2020)
ETC (Ainslie et al., 2020)
Big Bird (Zaheer et al., 2020)
Learnable Patterns
Low-Rank Transformer (Winata et al., 2020)
Longformer (Beltagy et al., 2020)
Swin Transformer (Liu et al., 2020)
Clustered Attention (Vyas et al., 2020)
Sinkhorn Transformer (Tay et al., 2020b)
Linformer (Wang et al., 2020b)
Low Rank / Kernels
Long Short Transformer (Zhu et al., 2021)
Fixed/Factorized/ Random Patterns
Adaptive Sparse Transformer (Correia et al., 2019)
Synthesizer (Tay et al., 2020a)
Random Feature Attention (Peng et al., 2021)
Blockwise Transformer (Qiu et al., 2019)
CC-Net (Huang et al., 2018)
GShard (Lepikhin et al., 2020)
Sparse
GLaM (Du et al., 2021)
Linear Transformer (Katharopoulos et al., 2020)
Image Transformer (Parmar et al., 2018)
Sparse Transformer (Child et al., 2019)
Switch Transformer (Fedus et al., 2021)
Product Key Memory (Lample et al., 2019)
Axial Transformer (Ho et al., 2019)
Scaling Transformer (Jaszczur et al., 2021)

Figure 2: Taxonomy of Efficient Transformer Architectures.