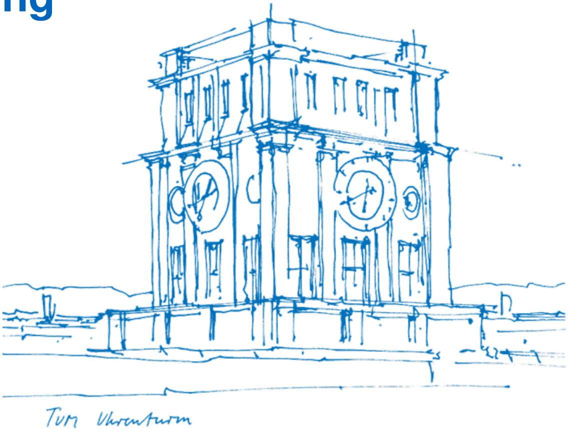
Computationally Efficient Architectures for Natural Language Processing

The Transformer and its Competitors

Kilian Brickl, Roman Alyaev

Chair of Scientific Computing in Computer Science
TUM School of CIT
Technical University of Munich

January 15th, 2025



General NLP Overview

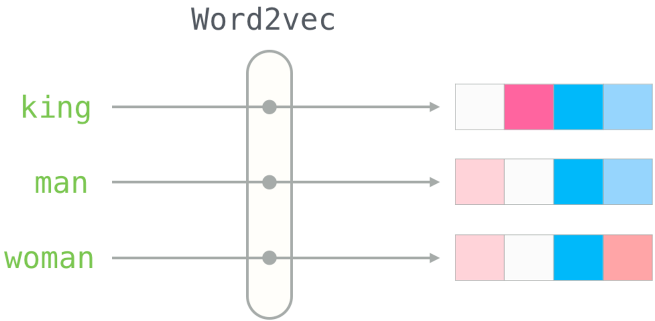
- → NLP bridges computer science, artificial intelligence, and linguistics.
- → Goal: Enable computers to understand, interpret, and generate human language.
- → Early methods relied on hand-crafted rules:
  - Examples: ELIZA and SHRDLU.
  - Challenges: Scalability and encoding linguistic complexity.
- → Shift to machine learning introduced statistical and deep learning models:
  - Enabled hierarchical data representation.
  - Expanded applications and improved performance.
- → Modern focus: Scalability, efficiency, and understanding long sequences.

Part I

Word Embeddings and Classical Architectures

Word Embeddings

- → Represents words as dense vectors.
- → Encodes semantic and syntactic meaning.
- → Learned from context in large text corpora.
- → Example: Word2Vec.



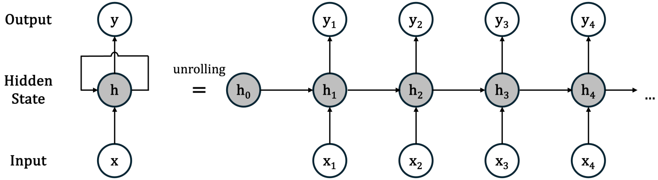
Word embeddings in vector space, showing relationships like "king - man + woman ≈ queen".

Classical Architectures: Overview

- → Predecessors of modern NLP models.
- → Recurrent Neural Networks (RNNs):
  - ☐ Sequential processing.
  - ☐ Captures dependencies over time.
- → Convolutional Neural Networks (CNNs):
  - ☐ Captures local dependencies in text.
  - ☐ Parallel computation for efficiency.

Recurrent Neural Networks (RNNs)

- → Designed for sequential data.
- → Maintains hidden states to capture sequence history.
- → Strengths:
  - ☐ Autoregressive generation of text.
  - ☐ Models short-term dependencies well.
- → Weaknesses:
  - ☐ Vanishing/exploding gradient problem.
  - ☐ Inefficient for long-range dependencies.



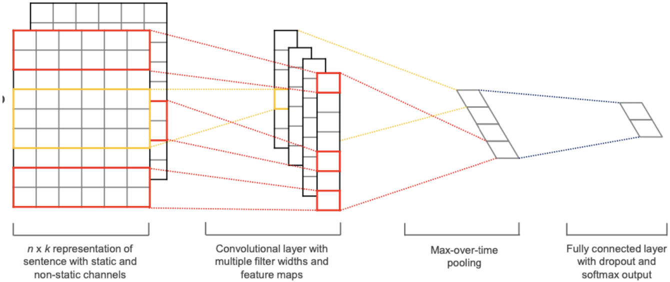
Convolutional Neural Networks (CNNs)

→ Adapted from computer vision for NLP.

→ Uses convolution operations on word embeddings.

→ Strengths:
  - ☐ Parallelizable, enabling faster computations.
  - ☐ Captures local features in text.

→ Weaknesses:
  - ☐ Limited by fixed receptive field.
  - ☐ Positional information not captured inherently.



Classical Architectures: Summary

→ RNNs and CNNs laid the groundwork for sequence modeling.

→ RNNs:
  - Sequential processing, good for short-term dependencies.
  - Struggles with long-range dependencies.

→ CNNs:
  - Efficient and parallelizable.
  - Needs additional mechanisms for positional information.

Part II

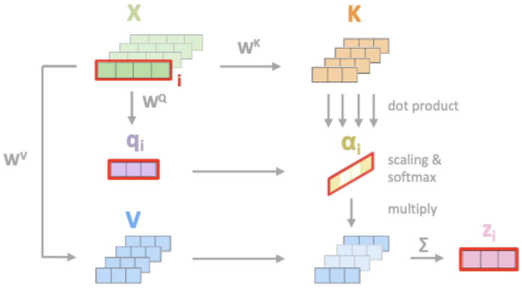
The Transformer Architecture

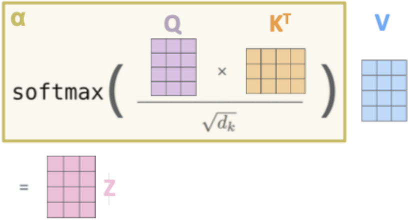
Transformer Overview

- → Does not rely on recurrence or convolutions:
  - ☐ Processes entire sequences in parallel.
  - ☐ Overcomes the sequential bottlenecks.
- → Key innovation is the attention mechanism:
  - ☐ Dynamically focuses on relevant parts of the input sequence.
  - ☐ Efficiently captures long-range dependencies.
- → Incorporates positional encodings:
  - ☐ Adds information about sequence order.

Intuition Behind Attention

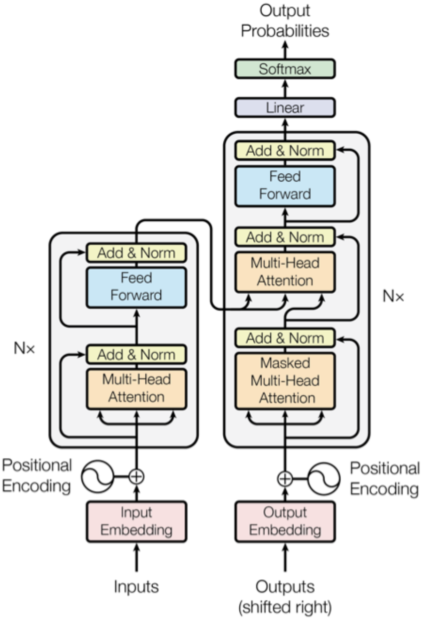
- → Attention identifies relevant parts of the input sequence for understanding each element.
- → Query (Q), Key (K), Value (V) matrices:
  - ☐ Query: Represents the element we want to understand.
  - ☐ Key: Determines which other elements are relevant to the Query.
  - ☐ Value: Contains the actual information from relevant elements.





Transformer Architecture

- → Encoder-decoder structure.
- → Key components of each block:
  - ☐ Multi-head self-attention.
  - ☐ Feedforward neural network (FFN).
- → Residual connections and layer normalization.
- → Encoder-decoder attention in the decoder.



Positional Encoding in Transformers

→ Transformers lack inherent sequence order awareness.

→ Positional encodings added to embeddings to retain sequence information.

→ Uses sine and cosine functions at varying frequencies:

$$PE(pos, 2i) = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right), \quad PE(pos, 2i+1) = \cos\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

→ Positional encodings allow the model to distinguish word order in a sentence.

Transformer Limitations

→ High Computational Complexity:

- Self-attention scales quadratically with sequence length $O(n^2 \cdot d)$.

→ Memory Requirements:

- Quadratic memory usage limits efficiency for long sequences.

→ Data Hungry:

- Requires large datasets for effective training.

→ Interpretability Challenges:

- Complex attention patterns make decision-making hard to interpret.

Part III

Competitors of the Transformer

Reformer

Overview

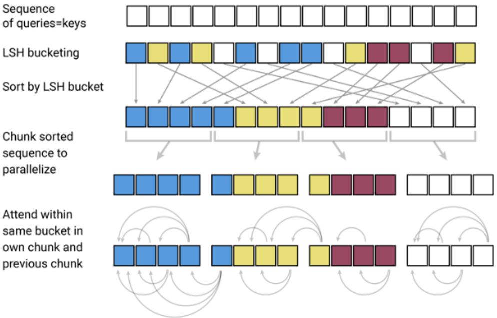
→ Key Features:
- Locality-Sensitive Hashing (LSH) attention reduces complexity to $O(n \log n)$.
- Reversible residual layers minimize memory usage.

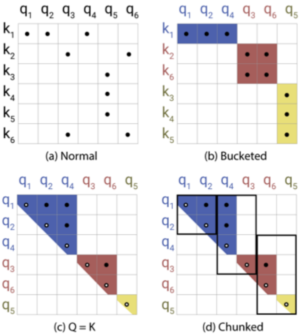
→ LSH Attention:
- Groups similar queries and keys into buckets.
- Attention computed only within each bucket.

→ Advantages:
- Efficient handling of long sequences.
- Lower memory footprint compared to vanilla Transformer.

LSH Attention

- → Reduces attention complexity from $O(n^2)$ to $O(n \log n)$.
- → Attention computed only within buckets:
  - ☐ Limits comparisons to similar tokens.
  - ☐ Avoids processing irrelevant pairs.



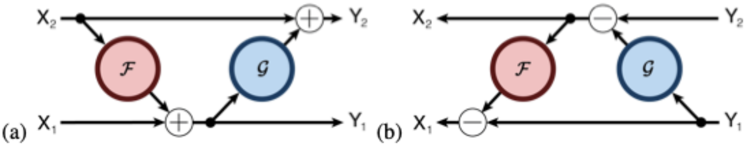


Reversible Residual Layers

- → Vanilla activation:

$$y = x + \text{Layer}(x)$$

- → Eliminates the need to store intermediate activations.
- → Reconstruction during backpropagation:
  - Activations are recalculated on-the-fly.



$$y_1 = x_1 + F(x_2)$$

$$y_2 = x_2 + G(y_1)$$

Performer

Overview

→ Key Features:
- ☐ FAVOR+ (Fast Attention Via Orthogonal Random features) approximates softmax attention.
- ☐ Reduces complexity to $O(n)$ in both time and space.

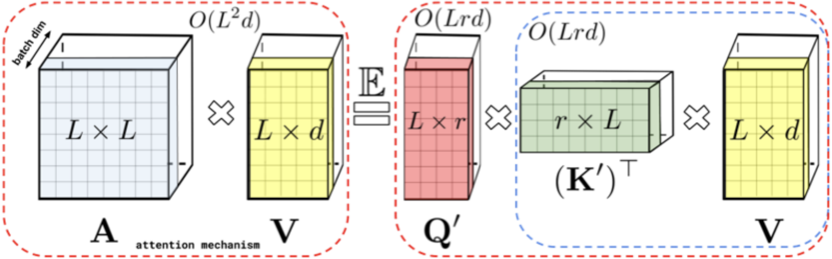
→ Linear Attention:
- ☐ Replaces traditional dot-product attention with kernel-based approximations.
- ☐ Efficient handling of extremely long sequences.

→ Advantages:
- ☐ Scales effectively for real-time tasks.
- ☐ Retains competitive performance with softmax attention.

Linear Attention (FAVOR+)

→ Approximates softmax attention using kernel-based methods.

→ Complexity reduced to $O(n)$:
- ☐ Avoids full pairwise comparisons.
- ☐ Suitable for very long sequences.



Linear Attention (FAVOR+)

→ Approximates softmax attention using random feature maps:

$$\text{Attention}(Q, K, V) = \Phi(Q)(\Phi(K)^T V),$$

where $\Phi(x)$ is a kernel-based feature map.

→ Complexity reduced to $O(n)$ in both time and memory:

- Avoids explicit computation of QK^T.
- Enables efficient handling of extremely long sequences.

→ Feature map example:

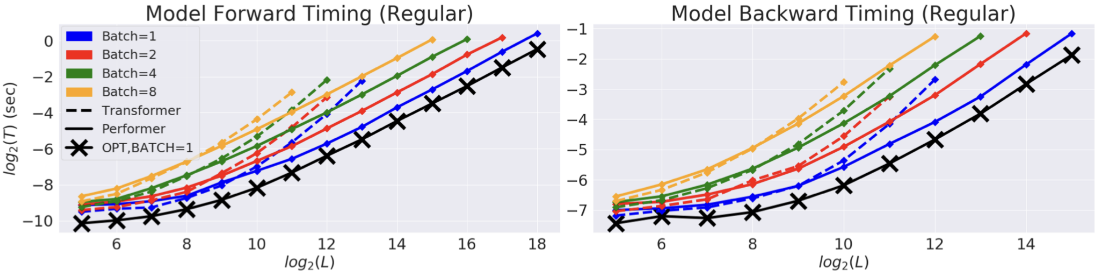
$$\Phi(x) = \exp\left(-\frac{\|x\|^2}{2}\right)\exp(\omega^T x),$$

where ω is a random vector from a specific distribution.

→ Normalization:

$$\text{Attention}(Q, K, V) = \frac{\Phi(Q)(\Phi(K)^T V)}{\Phi(Q)(\Phi(K)^T 1)}.$$

Performer Performance



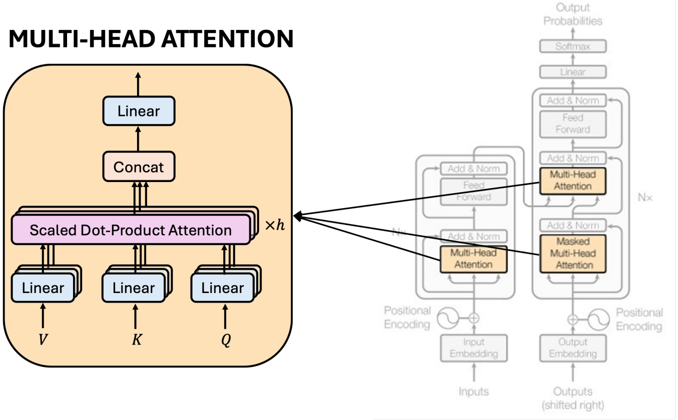
Performance of Performer on long-sequence tasks.

→ Practical for real-world applications:

- ☐ Low latency and high efficiency.
- ☐ Integrates easily with existing Transformer architectures.

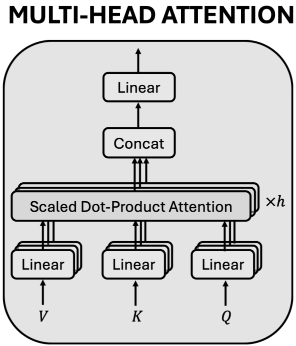
Attention-Free Transformer

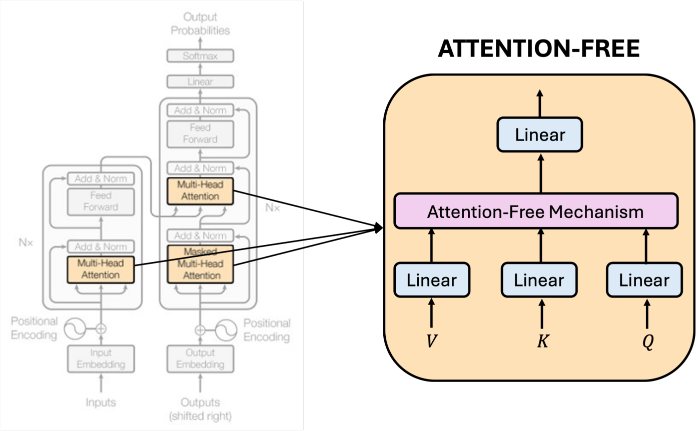
Attention-Free Transformer
Resolving the Quadratic Scaling Problem (1)



Attention-Free Transformer

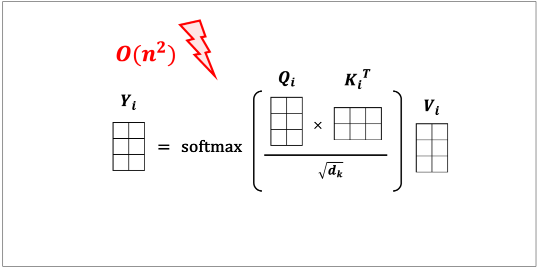
Resolving the Quadratic Scaling Problem (1)





Attention-Free Transformer
Resolving the Quadratic Scaling Problem (2)

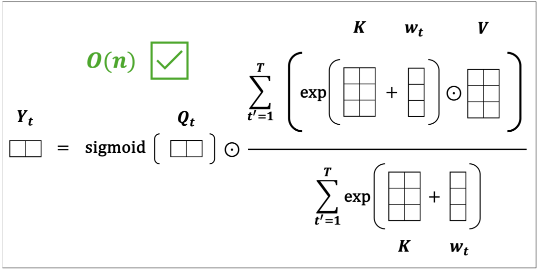
Multi-Head Attention:



→ Dot-Product between query and key

→ Concatenation over "dimensions"

Attention Free:



→ Element-wise multiplication between Q and weighted average of values

→ Concatenation over "time"

Attention-Free Transformer
Advantages and Disadvantages

Advantages

- ✓ Enables handling long inputs where traditional transformer may fail
- ✓ Retains ability to capture long-range dependencies
- ✓ Performance competitively with traditional transformers on many benchmarks

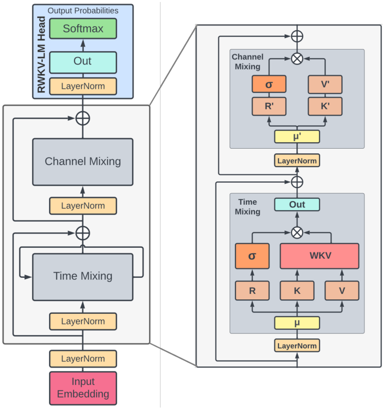
Disadvantages

- ✗ Not suited for tasks requiring fine-grained token-to-token interactions
- ✗ Not tested on real-world applications with huge amount of data and billions of parameters

Receptance Weighted Key Value (RWKV)

Receptance Weighted Key Value (RWKV)
Combining Transformers with Recurrent Neural Networks

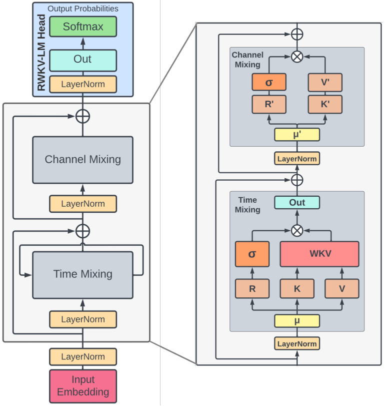
RKWV Residual Block



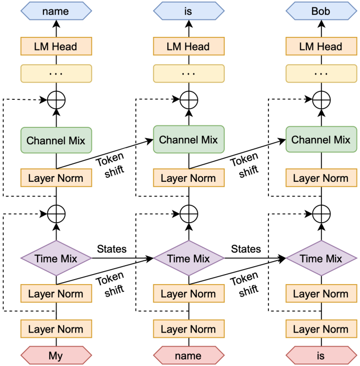
Receptance Weighted Key Value (RWKV)

Combining Transformers with Recurrent Neural Networks

RKWV Residual Block



Unrolling over time



Receptance Weighted Key Value (RWKV)

Recurrence - Parallelism Duality in the Time-Mixing Block

Parallel Formulation for Training

$$\text{WKV} = \frac{\sum_{i=1}^{T} \exp(k_i - (T-i)w)v_i}{\sum_{i=1}^{T} \exp(k_i - (T-i)w)}$$

Recurrent Formulation for Inference

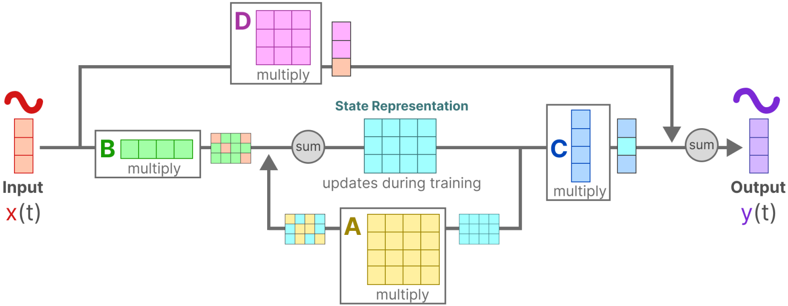
$$\text{WKV}_t = \frac{b_t}{a_t}$$

$$a_t = \exp(w)a_{t-1} + \exp(k_t)$$

$$b_t = \exp(w)b_{t-1} + \exp(k_t)v_t$$

State Space Models

State Space Models
From Control Theory to Natural Language Processing



$$h_t = Ah_{t-1} + Bx_t$$

$$y_t = Ch_t + Dx_t$$

State Space Models

Recurrence - Parallelism Duality in State Space Models

Unrolling the recurrence over time leads to

$$h_0 = Bx_0, \quad h_1 = ABx_0 + Bx_1, \quad h_2 = A^2Bx_0 + ABx_1 + Bx_2, \quad \ldots$$

and plugging h_t into $y_t = Ch_t$ gives

$$y_0 = CBx_0, \quad y_1 = CABx_0 + CBx_1, \quad y_2 = CA^2Bx_0 + CABx_1 + CBx_2, \quad \ldots$$

State Space Models

Recurrence - Parallelism Duality in State Space Models

Unrolling the recurrence over time leads to

$$h_0 = Bx_0, \quad h_1 = ABx_0 + Bx_1, \quad h_2 = A^2Bx_0 + ABx_1 + Bx_2, \quad \ldots$$

and plugging h_t into $y_t = Ch_t$ gives

$$y_0 = CBx_0, \quad y_1 = CABx_0 + CBx_1, \quad y_2 = CA^2Bx_0 + CABx_1 + CBx_2, \quad \ldots$$

This can be turned into the convolution

$$y = K * x$$

with the convolution kernel $K \in \mathbb{R}^L$ defined by

$$K = (CB, CAB, \ldots, CA^{L-1}B).$$

State Space Models

Extensions of the State Space Model

- **Structured State Space Models for Sequences (S4)** using the HiPPO matrix as system matrix
- **Structured State Space Models for Sequences with a Scan (S5)** to make the model simpler and more compact for multi-dimensional input
- **Selective and Structured State Space Models for Sequences with a Scan (S6)** to make the model able to focus more on relevant data, especially for text sequences

Mamba is a new model based on S6 blocks. It is a potential competitor to the transformer as it leverages parallel training and auto-regressive inference with a decent performance.

Part IV

Summary

Overview of Runtime and Space Complexities

Model	Time	Space
Transformer	$O(n^2 d)$	$O(n^2 + nd)$
Reformer	$O(n \log nd)$	$O(n \log n + nd)$
Performer	$O(nd^2 \log d)$	$O(nd \log d + d^2 \log d)$
AFT	$O(n^2 d)$	$O(nd)$
RWKV	$O(nd)$	$O(d)$
Mamba	$O(nd)$	$O(d)$

n is the sequence length and d the embedding dimension.