MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Yorsh Uladzislau** Personal ID number: **466899**

Faculty / Institute: **Faculty of Information Technology**

Department / Institute:

Study program: **Informatics**

Specialisation: **Knowledge Engineering**

II. Master’s thesis details

Master’s thesis title in English:

The Study of Linear Self-Attention Mechanism in Transformer

Master’s thesis title in Czech:

Studium lineárního self-attention mechanismu v transformerech

Guidelines:

Bibliography / sources:

Name and workplace of master’s thesis supervisor:

Mgr. Alexander Kovalenko, Ph.D. Department of Applied Mathematics FIT

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **07.04.2022** Deadline for master's thesis submission: _____

Assignment valid until: _____

Mgr. Alexander Kovalenko, Ph.D. Supervisor’s signature	Head of department’s signature	doc. RNDr. Ing. Marcel Jiřina, Ph.D. Dean’s signature

III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student’s signature

Bachelor's thesis

THE STUDY OF LINEAR SELF-ATTENTION MECHANISM IN TRANSFORMER

Bc. Uladzislau Yorsh

Faculty of Information Technology
Department of Applied Mathematics
Supervisor: Mgr. Alexander Kovalenko, Ph.D.
May 3, 2022

Contents

Acknowledgments vii

Declaration viii

Abstrakt ix

Abbreviations x

1 Introduction 1
- 1.1 A recurrent approach . . . 1
  - 1.1.1 Vanishing and exploding gradients . . . 2
  - 1.1.2 Gated networks . . . 2
- 1.2 The seq2seq architecture . . . 3
- 1.3 Attention . . . 4
- 1.4 Attention is all you need . . . 5

2 Transformer 7
- 2.1 Architecture details . . . 8
  - 2.1.1 A high-level view . . . 8
  - 2.1.2 Attention . . . 9
  - 2.1.3 Interpretability . . . 12
- 2.2 Application . . . 12
  - 2.2.1 Natural Language Processing . . . 13
  - 2.2.2 Computer Vision . . . 14
- 2.3 Analysis . . . 15
  - 2.3.1 Learning and transfer capabilities . . . 15
  - 2.3.2 Theoretical studies . . . 16
  - 2.3.3 Computational complexity . . . 17

3 Efficient Transformers 19
- 3.1 Sparse attention . . . 20
- 3.2 Memory-based approaches . . . 21
  - 3.2.1 Star Transformer and ETC . . . 21
  - 3.2.2 Set Transformer . . . 22
  - 3.2.3 Longformer . . . 22
  - 3.2.4 BigBird . . . 23
  - 3.2.5 Poolingformer . . . 23
  - 3.2.6 Luna . . . 24
- 3.3 Low-rank approaches . . . 24
  - 3.3.1 Linformer . . . 24
  - 3.3.2 Nyströmformer . . . 25
  - 3.3.3 Long-Short Transformer . . . 26
  - 3.3.4 H-Transformer-1D . . . 27
- 3.4 Recurrent approaches . . . 28

3.4.1 Transformer-XL . . . 28
3.4.2 Block-Recurrent Transformer . . . 28
3.5 Kernelized approaches . . . 29
3.5.1 (Kernelized) Transformers Are RNNs . . . 30
3.5.2 Performer . . . 30
3.5.3 Learning the kernel . . . 31
3.6 Analysis . . . 32

4 Proposed Variants **35**
4.1 Feedforward Kernel . . . 35
4.1.1 Architectures . . . 36
4.2 SimpleTRON . . . 37
4.3 Experiments . . . 38
4.3.1 Setup . . . 39
4.3.2 Results . . . 40
4.3.3 Discussion . . . 42

5 Summary **47**
5.1 A summary on modern efficient Transformers . . . 47
5.2 A summary on the proposed models . . . 48
5.3 On research trends and perspectives . . . 49
5.4 An impact of efficient architectures . . . 49

A Training Hyperparameters **51**

Attached Medium **59**

List of Figures

1.1 A comparison of RNN vs LSTM units. Converging lines denote concatenation, diverging denote copying, dotted lines separate individual LSTM gates, rectangles denote a linear layer with a corresponding activation function, circles denote an element-wise operation. . . . 3
1.2 The seq2seq architecture. Green resp. blue rectangles denote an encoder resp. a decoder. The RNN is pictured in its unrolled form. . . . 3
1.3 An attention-augmented seq2seq model. A fully-differentiable attention module dynamically weights input token annotations in a data-driven way, and creates a new vector of input sequence information for better decoding. . . . 4

2.1 The Transformer architecture. . . . 8
2.2 An autoregressive prediction example . . . 10
2.3 Attention operation described by[7] . . . 10
2.4 A scheme of Transformer attention matrices on an example of a half-done CZ-ENG translation. We employ a color scheme from the previous chapter and denote an encoder as green and a decoder as blue. White elements on the scheme denote zero elements (masked), while darker elements are used to emphasize larger alignment scores given by a model. . . . 11
2.5 Attention scores visualisation . . . 12
2.6 The ViT architecture. . . . 14
2.7 The DETR architecture. . . . 15

3.1 Sparse attention matrix schemes. . . . 20
3.2 Sparse + memory attention schemes. We employ different color schemes to denote global-to-global, local-to-local and mixed variants of an attention on the ETC figure. The ETC input is masked with certain global-to-local and local-to-local masks to reflect its structure. We also denote with pink the input tokens designated to global attention in Longformer. . . . 22
3.3 Luna encoder block. X denotes an input sequence, P denotes an auxiliary sequence. 24
3.4 Transformer-LS attention illustrated. L stands for an input length, w stands for a local attention span, B stands for blocks count, k stands for block size . . . 26
3.5 H-Attention illustration. N_r defines a (diagonal) block size in $A^{(l)}$. . . . 27
3.6 Block-Recurrent Transformer recurrent unit. K_e, V_e and K_s, V_s are shared for both paths. . . . 29
3.7 Kernel matrix of an attention operation, assuming that both queries and keys have length L. . . . 29

4.1 Transformer and SimpleTRON attention pipelines. Rejecting the softmax allows to use the matrix associativity to compute a compressed input representation to update queries. Matrices about to perform a product are emphasized. . . . 37
4.2 Scheme of the encoder block we are considering on this work. The additional skip connection is denoted with red. . . . 38
4.3 SimpleTRON example training plots. . . . 43

4.4 Example reconstructions of attention matrices for a 4001×4001 input of the BPE classification task. The value range is approximately $[0 - 0.005)$ for the left matrix and $(-150; 150)$ for the right. The right matrix was extracted as unnormalized (without the $1/\sqrt{L}$ term). 43
4.5 Training plot of the original Transformer fine-tuning on the LRA text classification task. Initial weights are given by the trained SimpleTRON model. 44
4.6 Training evolution of standard deviation of attention outputs. We compare the three models — vanilla Transformer, Simple and Simple-L. 44

List of Tables

1.1 Popular attention weighting function choices with sources. v and W denote learnable parameters, d_h denotes argument dimensionality. 5

3.1 Computational complexities of the reviewed attention mechanisms. The "-" instead of the learnable kernel complexity indicate that there are multiple models with different complexities. 34

4.1 Baseline and proposed models on the three LRA tasks. We denote sequence length as L, attention span as K and Sinkhorn model block size as B. C is an architecture-dependent multiplier explained in 4.3.2.1. 41
4.2 AG News fine-tuning results. 42
4.3 SimpleTRON training from-scratch results on AG News, with respect to the number of layers of the model. We denote fine-tuning results with an asterisk. 42

A.1 Hyperparameters used for the LRA experiments. 51
A.2 Hyperparameters used for the AG experiments. "Coarse" stage is tuning only attention weights, output layers and embedddings. "Fine" is a subsequent tuning of the whole model. 52

Throughout the writing of this thesis I have received a great deal of support and assistance.

I would first like to thank my supervisor, Mgr. Alexander Kovalenko, Ph.D., who guided me during my initial research and whose advice and patience tremendously contributed to the completion of this work.

I want to extend my sincere thanks to doc. Ing. Pavel Kordík, Ph.D., who actively participated in all related research and whose efforts to advance it cannot be underestimated.

I also gratefully acknowledge the assistance of Ing. Vojtěch Vančura, whose help made it possible to complement the work with a larger amount of experiments.

Finally, I wish to thank the CTU FIT faculty staff for being open and ready to help throughout all the six years of my study.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis. I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 3, 2022 .....

Abstrakt

Vzhledem k tomu, že kvadratická složitost mechanizmu vnímaní architektury Transformer způsobuje velké náklady na zpracování dlouhých posloupností, cílem dané práce je prozkoumat lineární varianty architektury a implementovat několik nových metod.

Klíčová slova hluboké učení, zpracování přirozeného jazyka, transformer, vnímaní, neuronové sítě

Abstract

As the quadratic complexity of an attention mechanism in the Transformer architecture places a high demand on processing long sequences, the goal of this research is to explore possibilities of linear attention in Transformer-like architecture and implement new methods.

Keywords deep learning, natural language processing, transformer, attention, neural networks

Abbreviations

BERT	Bidirectional Encoder Representations from Transformers
CV	Computer Vision
DETR	Detection Transformer
DGM	Deep Generative Model
DL	Deep Learning
EOS	End of Sequence
FAVOR+	Fast Attention via Orthogonal Positive Features
FCNN	Fully-Connected Neural Network
FPN	Feature Pyramid Network
GMM	Gaussian Mixture Model
GPT	Generative Pre-trained Transformer
LM	Language Model
LSTM	Long Short-Term Memory
ML	Machine Learning
MHA	Multi-Head Attention
MLM	Masked Language Modeling
NLP	Natural Language Processing
NN	Neural Network
NSP	Next Sentence Prediction
PRF	Positive Random Features
R-CNN	Region-Based CNN
RKS	Random Kitchen Sink
RNN	Recurrent Neural Network
SOS	Start of Sequence
T5	Text-to-Text Transfer Transformer
TE	Text Entailment
ViT	Vision Transformer

Chapter 1
Introduction

In this chapter we will briefly introduce the motivation standing behind the Transformer architecture. We will quickly review RNN architectures used for machine translation, explain difficulties corresponding to application of recurrent models in practice and will take a view on the most advanced pre-Transformer sequence processing network. At the end of the chapter, we will introduce an attention mechanism with an explanation how it may improve a recurrent model, and eventually will arrive to an architecture completely built around an attention.

Recurrent neural networks have undergone many improvements throughout their existence, which eventually led them to the state-of-the-art status for many sequence processing and transduction tasks. They offer a natural approach to sequence processing, based on reading an input sequence one element at time, and outputting a corresponding vector based on the current input and an internal state.

However, the path between a theoretical formulation and an actual practical application was long and full of engineering difficulties. While for many problems a cure was found, some fundamental recurrent model issues could not be solved. The research eventually led to a radically new architecture, which not only got rid of the shortcomings of the recurrent approach, but also turned out to be universal enough to conquer the status of the most powerful model in a plenty of domains.

In this chapter, we will approach the Transformer architecture by introducing preceding models. A deep dive into a neural NLP history is out of scope of this work; instead, we will introduce the state-of-the-art approaches "right before" an emergence of attention-based models with a necessary background on RNNs, and will illustrate attention principles and motivation on the transition between architectures.

1.1 A recurrent approach

A Recurrent Neural Network is an architecture which relies both on input vector and an internal state to produce an output. A hidden state is a key difference from FCNNs, which allows the cell to operate differently after previously seeing different inputs, and thus to capture time-dependent relations between input elements.

A RNN cell output may be generally written as a function of an input vector, its internal state and its parameters:

$$\begin{aligned} \mathrm{RNN}(x_t) = h_t &= f(x_t, s_{t-1}; \theta) \\ s_t &= g(x_t, s_{t-1}; \theta) \end{aligned}$$

which is in the most simple case a linear-and-activation relationship:

$$h_t = \sigma(W[x_t, s_{t-1}] + b)$$
$$s_t = h_t$$

where σ is a non-linear activation function, s_0 resp. s_t are hidden states after initialization resp. after step t, $[\cdot, \cdot]$ is a concatenation along a hidden dimension axis and $(W, b) = \theta$. A RNN cell defined this way is already capable of learning complex dependencies, but suffers a variety of problems related to its sequential nature.

1.1.1 Vanishing and exploding gradients

A single RNN cell may be unrolled to a pretty deep neural network, which applies a similar transformation multiple times to the same input. For simplicity, we can approximate a recurrent relationship with a simple linear transform:

$$s_t \approx W_s s_{t-1}$$

Assuming a network initialization with small weights and a zero-centered input, the approximation is fair for the most common choices of activation functions, namely logistic sigmoid or tanh. After performing an eigenvalue decomposition of $W_s = Q\Lambda Q^{-1}$ and a repeated application of this transformation on s_0 for t times, we obtain:

$$s_t = Q\Lambda^t Q^{-1} s_0$$

The final result is determined by the spectral radius $\lambda = \rho(W_s)$. If $\lambda > 1$, hidden state vectors will grow in magnitude and eventually will destabilize the training. If $\lambda < 1$, than s_t will tend to zero and the training will be impeded. It is also true for both cases, that a small change of W_s will result in an exponential impact on s_t values.

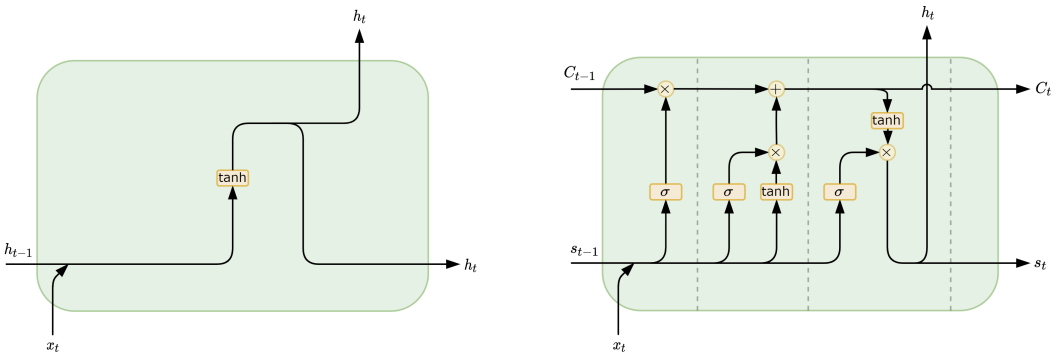
The problem of a potential gradient explosion limits the choice of activation functions to bounded ones. And since bounded functions are saturating, they are prone to significantly reduce the gradient passing backwards during training, and thus make handling long inputs intractable.

1.1.2 Gated networks

To address the issue, a RNN cell state may be updated in a more subtle way, without a major recomputation of its values. This may be achieved via gating, and one of the most known realisations of a gated recurrent network is the Long Short-Term Memory[1] (LSTM).

A LSTM unit detaches a cell state C_{t-1} from a preceding output h_{t-1}, and leverages five different linear layers grouped into three gates, which determine what the cell needs to *forget* and what it needs to bring along to the next step. Further in this subsection, we will refer a $[x_t, h_{t-1}]$ concatenation as *a cell input*, passing some input through a tanh-activated layer as *a transform*, and a $f(x, y) = x \odot \sigma(Wy + b)$ operation as a *gating x with y*, where σ is a logistic function.

- A **forget** gate gates a cell state with a cell input.
- An **input gate** gates a cell input with a transformed cell input, and then adds a result to the cell state.



(a) RNN **(b)** LSTM

Figure 1.1 A comparison of RNN vs LSTM units. Converging lines denote concatenation, diverging denote copying, dotted lines separate individual LSTM gates, rectangles denote a linear layer with a corresponding activation function, circles denote an element-wise operation.

- An **output gate** transforms a cell state and gates it with a cell input.

The LSTM architecture exploits an idea of ”forgetting” the unrelated information, and gets rid of its undesired fragments by gating. After combining this approach with a minimalist cell state update strategy, we obtain a RNN unit able to handle much longer sequences.

The LSTM had motivated a further research on gated recurrent networks[2], spawning thousands of possible unit layouts, but the basic LSTM units similar to the one described were pretty persistent in scientific publications, and served in the state-of-the-art models until Transformers emerged.

1.2 The seq2seq architecture

The basic RNN model formulation for sequence transduction supposes first to read an input sequence of t tokens to produce its vector representation h_t. Given that representation as a hidden state, the model should output members of another sequence one-by-one until a special ”end of sequence” token will be given.

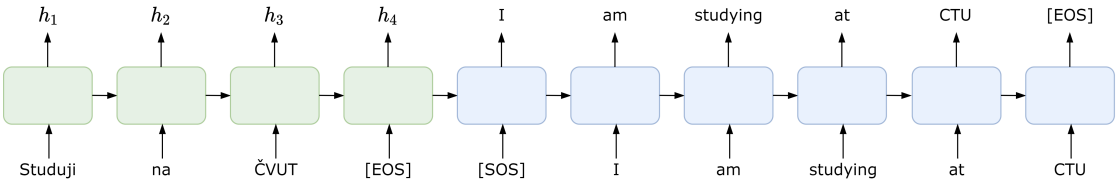


Figure 1.2 The seq2seq architecture. Green resp. blue rectangles denote an encoder resp. a decoder. The RNN is pictured in its unrolled form.

The Sequence-to-Sequence architecture[3] makes a one step further, and separates weights for embedding and generating, effectively turning the architecture into a LSTM-based autoencoder. This encoder-decoder approach is still the basis of sequence transduction architectures today.

1.3 Attention

Despite the success of a seq2seq architecture and its further improvements (bidirectional encoder, new recurrent units, new methods of training and regularization, ...), it still shared the common shortcoming of all recurrent models: the variable-sized input was represented by a fixed-size vector. The vector of a finite dimensionality had turned into a bottleneck, effectively preventing a model from capturing longer contexts.

Bahdanau et al.[4] proposed to keep encoder outputs (referred as *annotations* of input tokens) and aggregate them into a context vector for each decoded token. The output is then conditioned by a previously decoded token, a hidden state and additionally by this context vector:

$$\begin{aligned}
\text{AttDecoderRNN}(y_t) &= f(y_t, s_{t-1}, c_t; \theta) \\
s_t &= g(y_t, s_{t-1}, c_t; \theta) \\
c_t &= \sum_{i \in I_x} \alpha_i h_i \\
\alpha_i &= \frac{\exp(e_i)}{\sum_{j \in I_x} \exp(e_j)} \\
e_i &= \text{sim}(h_i, s_{t-1})
\end{aligned}$$

where I_x is an index set of input tokens x_i, and h_i are input annotations.

As we can see, a context vector is basically a convex combination of input annotations. The coefficients are given by values of some similarity (or *alignment*) function between a decoder hidden state and input annotations, normalized with a softmax. This allows to "attend" to relevant pieces of an input sequence and to use this information for better decoding, while also improving a gradient flow through the model.

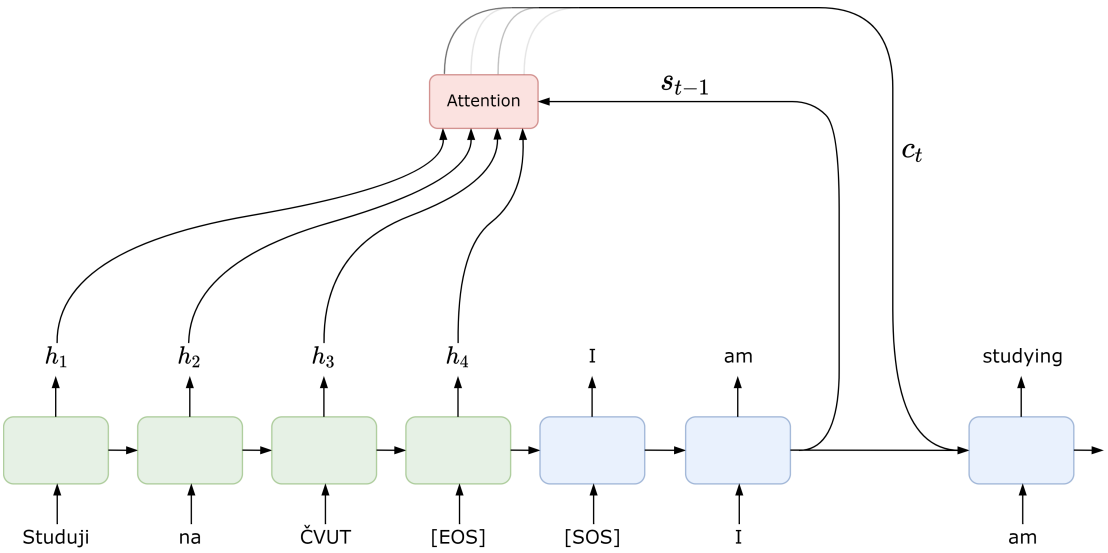


Figure 1.3 An attention-augmented seq2seq model. A fully-differentiable attention module dynamically weights input token annotations in a data-driven way, and creates a new vector of input sequence information for better decoding.

Bahdanau et al. proposed the following rule to compute a similarity function:

$$\text{sim}(s_{t-1}, h_i) = v_a^T \tanh(W_a[s_{t-1}, h_i])$$

However, impressive results shown by the model motivated a further research, and quickly spawned numerous variants of attention computation, each with their pros and cons. Since describing all attention mechanisms is out of scope of this work, we will restrict ourselves to a table containing several popular variants:

Name	$\text{sim}(h_i, s_{t-1})$ definition	Source	Note
Additive/concat	$v^T \tanh(W[h_i, s_{t-1}])$	Bahdanau et al.[4]	-
Content-based	$\text{cosine}(h_i, s_{t-1})$	Graves et al.[5]	A Neural Turing Machine uses this attention to address external memory.
Location-based	Ws_{t-1}	Luong et al.[6]	Weighting is performed only by target location.
General	$h_i^T W s_{t-1}$	Luong et al.[6]	-
Dot	$h_i^T s_{t-1}$	Luong et al.[6]	-
Scaled Dot	$h_i^T s_{t-1}/\sqrt{d_h}$	Vaswani et al.[7]	Performs scaling to avoid softmax saturation.

Table 1.1 Popular attention weighting function choices with sources. v and W denote learnable parameters, d_h denotes argument dimensionality.

1.4 Attention is all you need

An attention mechanism let recurrent neural networks to strengthen their positions in machine translation, but the essentially sequential nature precluded training parallelization. Furthermore, an attention allowed to process longer sequences but did not solve the context forgetting problem completely. After the work on these problems, Vaswani et al.[7] have found the solution in the architecture completely free of recurrence, and relying only on attention and linear layers to produce new input representations. The detailed description of the architecture is the subject of the next chapter.

Chapter 2
Transformer

In this chapter, we will explore the Transformer architecture in detail. We will describe the attention operation, its information retrieval interpretation and a role of individual attention variants used in the model. We will also show that both encoder and decoder parts of the architecture can be used separately, and will introduce some model variants which have found their use in practice. Finally, we will explore the theoretical background and make some conclusions about the model operation.

The Transformer[7] is a prominent neural network architecture, which has already established itself as a state-of-the-art model in a vast amount of tasks. While NLP and sequence processing largely motivated its emergence, the architecture showed itself surprisingly modality-agnostic and became a state-of-the art in language modeling[8] and text generation[9][10][11][12], as well as in image classification[13] and object detection[14], speech recognition[15], reinforcement learning[16] and other domains.

The source of a power is an attention mechanism, which is based on pairwise matching of input elements to discover relationships. Such a framework allows the model to handle arbitrarily long input sequences, and to discover long-spanning dependencies without a need to squeeze all the information into a single vector. Furthermore, the pairwise matching allowed to process data of any modality, and eventually led to another way to look at data, as well as to a new approach of designing neural network architectures[17].

However, everything got its own cost. Being an unbiased architecture employing pairwise matching, Transformer also suffers from data inefficiency and high computational costs. Current research is largely motivated by overcoming the model issues, and we can note several primary directions of a work:

- *Domain adaptation.* Transformers require significantly more data than competing architectures to train. For the major part of tasks, it is needed to employ transfer learning tools or advanced domain adaptation techniques to utilize the knowledge already accumulated in large pretrained models.
- *Computational efficiency.* Eschewing a recurrence allows to heavily parallelize both inference and training, but the true issue lays in the asymptotic complexity. A pairwise matching framework induces both large memory and computation costs, reducing the benefit of no recurrence.
- *Generalization.* While most of tasks can be solved via transferring an already pretrained Transformer, these base models still need to be somehow trained. Advanced pretraining techniques and a heavy use of unlabeled data allow to improve the final model performance for future downstream tasks.

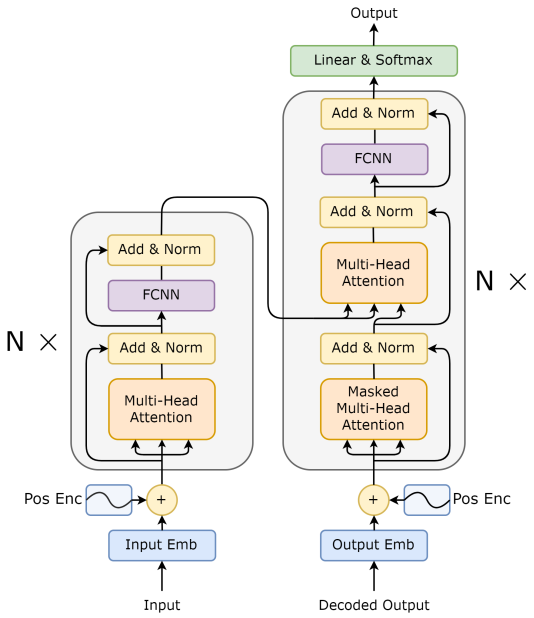


Figure 2.1 The Transformer architecture.

This chapter describes all the base architecture details needed to understand the main part of the work. The first section describes main building blocks, an information flow and individual attention variants with their designated roles in a transductive model. The second section discusses the use of individual model parts and introduces some architectures built upon Transformer encoder or decoder. The third section introduces the model application in various tasks without restriction to sequence processing only. Finally, we discuss the architecture features and peculiarities, introduce some theoretical research and problems which are being tackled by the models from the main part of the thesis.

2.1 Architecture details

2.1.1 A high-level view

The original model by[7], referred in the subsequent chapters as a *vanilla Transformer*, consists of an input layer, an encoder, a decoder and an output layer, see Fig. 2.1. At a high level, the model encoder processes an input sequence of L tokens into a $L \times d$ dense representation, progressively re-encoding each token using the context of all the sequence. Given this representation, a decoder autoregressively generates a new sequence, one token at a time.

2.1.1.1 Input layer

An input layer consists of an embedding layer and positional encodings:

$$\text{Input}(X) = \text{LayerNorm}(\text{Embedding}(X) + P), X \in N_0^L, P \in \mathbf{R}^{L \times d}$$

The layer borrows a lot from embedding layers of recurrent seq2seq architectures. However, the Transformer got no recurrence nor implicit ordering, and to bring in positional signal it is

needed to add special vectors to the input. Some architectures may further extend the layer with additional encodings, e.g. token types.

2.1.1.2 Encoder

An encoder is composed of multiple stacked encoder blocks, each of which consumes and produces an input sequence representation of a $L \times d$ shape. Each encoder block, in turn, consists of a **multi-head self-attention** layer and a FCNN applied token-wise. A multi-head self-attention layer exploits an idea of building a new token representation by gathering an information from all tokens in the sequence, while a token-wise FCNN performs an additional non-linear transform. There is a skip connection after each attention layer and a FCNN, followed by a Layer Normalization.

2.1.1.3 Decoder

Similarly to an encoder, a decoder is a stack of decoder blocks, preceded by an embedding layer. Each decoder block accepts a currently decoded sequence and an encoder output, and produces a new decoded sequence representation.

A decoder block is similar to an encoder block, but inserts an additional layer and uses different kinds of attention:

1. First, it performs a **masked self-attention**. We describe it more in detail in the next subsection, but the role of this layer is to compute new decoded tokens representations by aggregating an information from *previous* tokens, impelling an autoregressive behavior.
2. Then, it performs a **cross-attention** between a decoded sequence and an encoder output. This is how the model propagates an information from an input sequence to a decoded one.
3. Finally, a position-wise FCNN is applied the same way as in an encoder.

Identically to an encoder, a decoder makes use of residual connections and LayerNorms over each of the three layers.

2.1.1.4 Model output

An output layer is a simple token-wise linear layer with a softmax on top. In an autoregressive setting, the model is being trained via teacher forcing, while an inference is performed in the following way:

1. Encode an input sequence.
2. Feed a [SOS] token into a decoder.
3. Get the last generated token, append it to the decoded sequence and feed it back.
4. Repeat until a [EOS] token will be generated.

However, as we will see in the following section, the model can be trained for a remarkable variety of tasks, and can make use of other output formulations.

2.1.2 Attention

The key module of the Transformer architecture is an attention. We have briefly noted the roles of particular attention layers, but this subsection is dedicated to their detailed description, which will shed some light on how actually the model utilizes the context of a token to transform it.

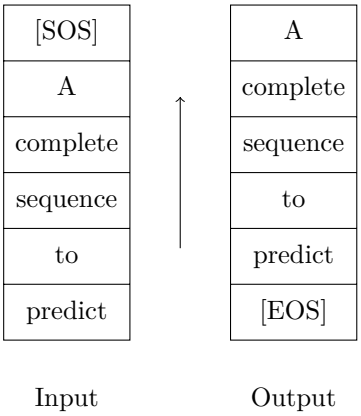


Figure 2.2 An autoregressive prediction example

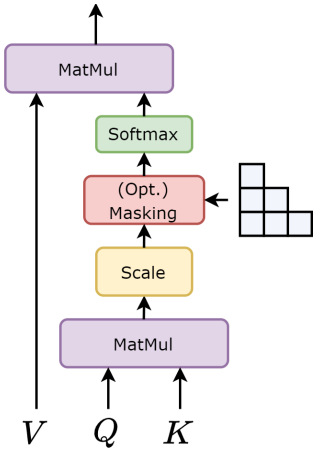
The authors in[7] use a specific terminology to define the operation. It may be seen as a function mapping **query**, **key** and **value** to an output. Output vectors are weighted sums of value vectors, where weights are given by some similarity function between query and key vectors.

More specifically, the authors define the attention operation this way: given queries and keys of dimension d_k and values of dimension d_v, stacked into matrices by order, compute the following:

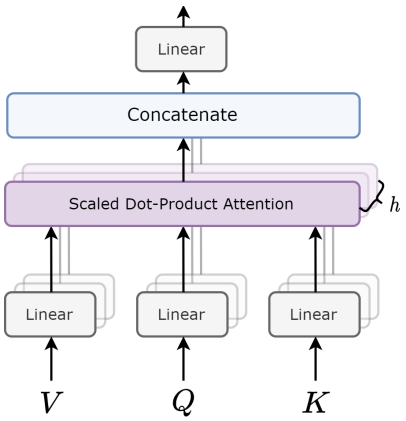
$$\text{Att}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

The $\sqrt{d_k}$ term is needed to avoid the softmax saturation, since a dot product of vectors with large d_k may produce large magnitudes and thus decrease a gradient flow.

Considering a query matrix of size $L_q \times d_k$, a key matrix of size $L_v \times d_k$ and a value matrix of size $L_v \times d_v$, we will refer to the intermediate result of size $L_q \times L_v$ after applying a softmax as an **attention matrix**.



(a) Scaled Dot-Product Attention



(b) Multi-Head Attention running several attentions in parallel

Figure 2.3 Attention operation described by[7]

The operation may be interpreted as a pairwise comparison of queries and keys, and a particular row of the matrix reflects how each key is related to the query vector q_i. By multiplying

a value matrix with this row, we form a new vector which will represent the i^{th} query token at the block output.

The Transformer architecture by[7] utilizes three slightly different attention operation variants (see Fig. 2.4) described below:

2.1.2.1 Self-Attention

During a Self-Attention operation query, key and value come from a single source – an input sequence, projected by three different linear layers. This operation is being performed in the encoder and is used to re-encode input tokens considering their relevance to other tokens.

2.1.2.2 Masked Self-Attention

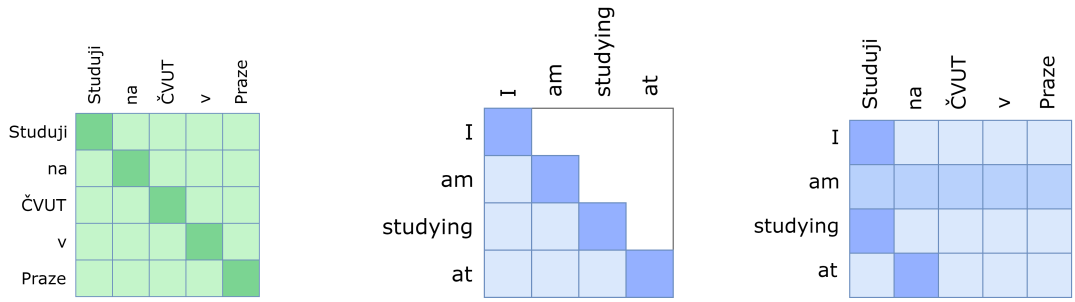
Masked Self-Attention works similarly to Self-Attention, but before applying a softmax the attention matrix is being masked. Considering a sequence beginning at the top/left of an attention matrix, a $-\infty$ is being added to the elements above the diagonal before applying a softmax, disallowing to attend "future" tokens by previous tokens in a sequence.

A Masked Self-Attention operation is used in the decoder. Masking an attention matrix this way forces an autoregressive behavior instead of a plain information copying on a next token prediction.

2.1.2.3 Cross-Attention

Cross-Attention performs an attention operation on queries and key-value pairs coming from different sources. The model by[7] performs an unmasked attention with queries being decoder inputs and keys+values being encoder outputs. This allows to "mix in" an information from an encoded sequence into a decoded one.

The three described attention modules are the most common and capable of solving most tasks, but the formulation of an attention allows further applications if any will be found useful, e.g. by varying Q, K and V sources or mask shapes.



(a) Self-Attention **(b)** Masked Self-Attention **(c)** Cross-Attention

Figure 2.4 A scheme of Transformer attention matrices on an example of a half-done CZ-ENG translation. We employ a color scheme from the previous chapter and denote an encoder as green and a decoder as blue. White elements on the scheme denote zero elements (masked), while darker elements are used to emphasize larger alignment scores given by a model.

2.1.2.4 Multiple Heads

The authors found it beneficial to project attention inputs h times with h different sets of query, key and value matrices. The results of h individual attention operations are then being

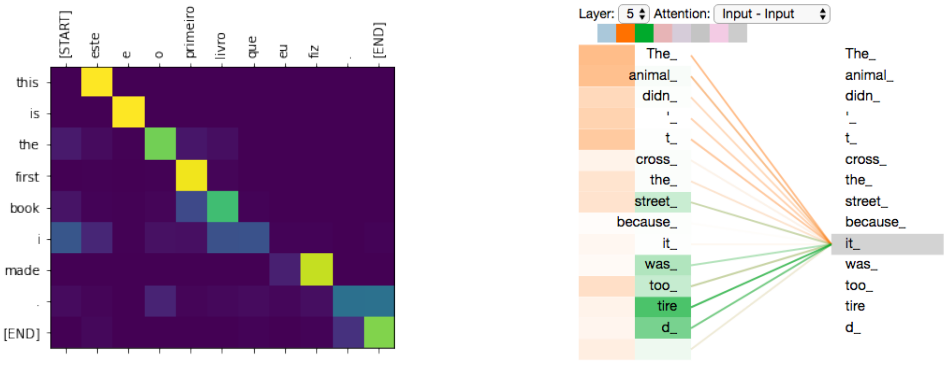
concatenated and projected by a linear layer of size $hd_v \times d_{hidden}$:

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= [\text{head}_1, \text{head}_2, \ldots, \text{head}_h]W^O \\ \text{head}_h &= \text{Att}(QW_i^Q, KW_i^K, VW_i^V) \end{aligned}$$

Where $W_i^Q \in \mathbf{R}^{d_{hidden} \times d_k}$, $W_i^K \in \mathbf{R}^{d_{hidden} \times d_k}$, $W_i^V \in \mathbf{R}^{d_{hidden} \times d_v}$ and $W_i^O \in \mathbf{R}^{hd_v \times d_{hidden}}$.

By using smaller values for d_k and d_v, the computational costs may be left similar to the single-head attention, but with a possibility to split individual transforms between GPUs. The another advantage is that individual heads may learn different things, since they apply different linear transforms before constructing a non-linear attention matrix, allowing to learn more complex functions. This can be seen as heeding different aspects of input, e.g. token semantics vs token morphology.

2.1.3 Interpretability



(a) Attention matrix visualization as a heatmap from [18].

(b) A per-head view on attention scores via BertViz tool[19].

Figure 2.5 Attention scores visualisation

An attention matrix produced by all attention models is a kind of a built-in interpretation tool. We have already mentioned that its scores represent a relevance between queries and keys, and visualizing them may give some insights about model reasoning. The two most common ways to visualize attention are matrix heatmaps and weighted bipartite graphs, see Fig. 2.5. In the case of Transformer, the latter may be more preferable since it allows to depict multiple attention heads at time.

2.2 Application

A word on operation modes. Sequence processing tasks are not limited to machine translation only; for tasks which are not built around a sequence transduction a usage of the encoder-decoder architecture may be excessive. That is why a substantial part of Transformer-based architectures is built over an encoder- or decoder- only.

Encoder-only models are useful for downstream tasks primarily including sequence classification or language understanding. These architectures typically leverage only self-attention, do not perform causal masking, and make use of bidirectional token context. Before fine-tuning

for downstream tasks, they are typically being pretrained on denoising and input consistency objectives.

Decoder-only models are meant for autoregressive tasks. Similarly to encoder-only models, they leverage only self-attention, but the key difference is the causal mask which does not allow to "peep into the future", and given some context or control sequences allows to generate new text. The common pretraining pipeline includes the next token prediction objective and teacher forcing.

Encoder-decoder models are meant for sequence-to-sequence tasks, such as machine translation, text summarization or generative question answering. They generally follow the original Transformer architecture introduced in the previous section, and may include more complex training objectives than encoder- or decoder- only models. Due to the nature of sequence-to-sequence tasks, encoder-decoder models typically require parallel corpora of training data, and thus are harder to implement in practice. However, a one may partially counter the problem by making use of unsupervised data and pretraining an encoder and decoder separately on corresponding objectives.

We demonstrate a variety of Transformer-based models by introducing several examples below.

2.2.1 Natural Language Processing

Transformers were originally designed for machine translation, and due to historical reasons most of works on them are inclined towards the NLP domain. The majority of NLP Transformers share the pretrain-and-tune approach for particular downstream task, but fill the model capacity with different pretraining strategies.

2.2.1.1 BERT

BERT[8] (**B**idirectional **E**ncoder **R**epresentations from **T**ransformers) is a transformer encoder pretrained on two tasks – masked language modeling (**MLM**) and next sentence prediction (**NSP**). The MLM task consists of replacing some input tokens with a special [MASK] token and training a model to predict them. The NSP task in turn consists of encoding two concatenated input sentences and predicting whether the second sentence follows the first.

This conceptually simple pretraining strategy allows the model to yield representative input embeddings, which unlike word2vec-based methods[20] also include contextual information. By fine-tuning the pretrained network, we can obtain a state-of-the-art model for sequence classification, question answering and other language understanding tasks.

The BERT architecture became ubiquitous in NLP pipelines, and its success largely motivated a further work on its pretraining strategies (RoBERTa[21], SpanBERT[22], ELECTRA[23]), distillation (DistillBERT[24], ALBERT[25]) and other directions. The model is still serving as a strong text processing baseline, as well as a foundation for new architectures.

2.2.1.2 GPT

GPT (**G**enerative **P**re-trained **T**ransformer)[9][10][11] is a decoder-based model family for text generation. There are currently three iterations of GPT models at the moment, all of which generally share the model architecture (stacked decoder blocks with a masked self-attention), and place an emphasis on language understanding through generative pretraining (in contrast to BERT), large capacity and zero-shot learning.

The **first generation of GPT** models[9] shars the general structure with BERT-Base. The pretraining pipeline is built over the next token prediction task, while during fine-tuning both the downstream objective and the generative loss are being optimized. The model also makes an

extensive use of task-specific input transformations, so that a structured input (like a premise-hypothesis for text entailment) can be fed as a contiguous text sequence separated by delimiter tokens. The model became a new state-of-the-art on 9 of 12 observed tasks and achieved a decent zero-shot accuracy on a variety of problems.

The **GPT-2**[10] demonstrates primarily a quantitative development rather than a qualitative. However, the authors discussed and experimented with two important concepts — task conditioning with a natural language sequence and zero-shot task transfer. The crucial part for implementing these two concepts is the huge model capacity, and that is why the second GPT model was roughly 10× larger than the first one, comprising 1.5B parameters.

The **GPT-3**[11] model is again a straight improvement of GPT-2 in terms of size and capacity. With its astounding size of 175B parameters, it lets to further explore capabilities of large language models. The authors discovered that LMs of such capacity are able to extract patterns from text and to exploit them at zero-shot tasks, yielding impressive results or even beating state-of-the art models without any tuning.

2.2.1.3 T5

T5[26], or **T**ext-**t**o-**T**ext **T**ransfer **T**ransformer, is an encoder-decoder architecture which is based on casting all the NLP tasks (including text classification, question answering and demasking) to sequence-to-sequence problems. This allows to leverage the same model and training objective for a variety of problems, and to make a use of multi-task pretraining.

Architecturally, the model is very similar to the vanilla Transformer, with some minor layer rearrangements and simplified relative positional encodings. The main difference is the output modality – instead of task-specific heads like softmax classifiers or scorers, the model output is always a text, corresponding to an input task, e.g. a class label, "negative/positive/non-TE" for entailment tasks or masked tokens for denoising objectives.

Since the main contribution of the work is not the model itself but rather a thorough study on objectives and training settings, we suggest the reader to refer it for more details. To conclude, we will mention that the model performed surprisingly well on all common fine-tuning benchmarks.

2.2.2 Computer Vision

2.2.2.1 ViT

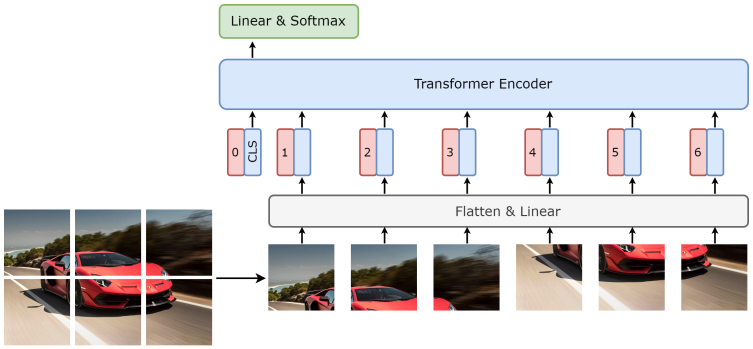


Figure 2.6 The ViT architecture.

The **Vi**sion **T**ransformer[13] architecture is one of the first successful attempts to apply Transformers in the CV domain. The model firstly cuts an input image into rectangular patches, which are then being linearly projected, arranged into a sequence and summed with absolute positional encodings. The rest of the processing pipeline is identical to the sequence classification.

Compared to previous efforts, ViT succeeded through extracting larger image patches (16×16 instead of per-pixel or 2×2) and an extensive pretraining – to achieve a state-of-the-art performance on the ImageNet-1k dataset, the model was trained on significantly larger datasets, such as ImageNet-21k or JFT-300M.

While the original ViT demonstrated a significant data inefficiency, recent works suggest remarkably more efficient training pipelines[27], which allow to improve the model performance and even outperform CNNs after training only on the ImageNet-1k dataset, thereby making ViT an image classification state-of-the-art architecture at the moment.

2.2.2.2 DETR

Detection **Tr**ansformer[14] is an object detection architecture, which rejects detection priors such as anchors and non-maximum suppression. Instead, it offers a more elegant task formulation as a set prediction using the Transformer encoder-decoder.

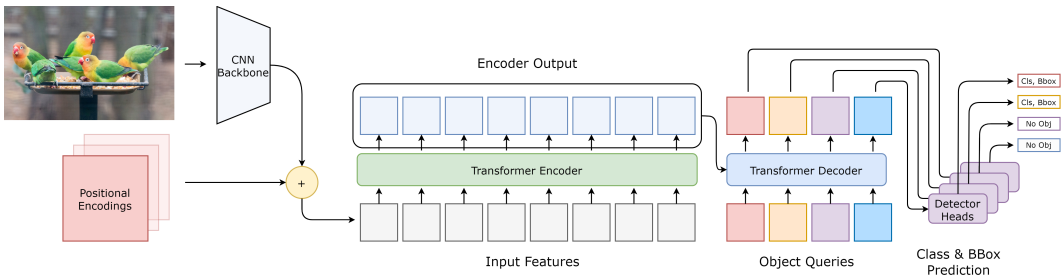


Figure 2.7 The DETR architecture.

The model embodies three components — a CNN backbone, a Transformer and a FCNN prediction head. The CNN produces a feature map of low spatial resolution, which is then being compressed along hidden dimension, summed with absolute positional encodings and fed as a sequence to an encoder. At the same time, a high amount of dummy tokens (*object queries*) is fed to a non-causal decoder. The model head predicts for each transformed object query a class (including a "no object" option) and a normalized bounding box. During training, the predicted objects are being dynamically matched to ground truth boxes via the Hungarian algorithm.

The model performance turned out to be competitive with a FPN-augmented Faster R-CNN. Interestingly, while the DETR underperformed on small objects detection, it was able to detect large objects more successfully due to the global nature of an attention, as opposed to the locality of convolutions.

2.3 Analysis

The architecture success in a variety of tasks has motivated a wide research on model capabilities, both experimental and theoretic. This section tries to summarize the most important findings along with the ones needed before moving on to the next chapter.

2.3.1 Learning and transfer capabilities

While Transformers are able to reach outstanding performance in majority of tasks, that is significantly conditioned by the amount of data for traing. Instead of stronger assumptions made by CNNs (equivariance in space, local correlation) or RNNs (larger relevance of the closest context), Transformers rely only on a dot-product similarity of input elements, thus eliminating locality bias and maintaining permutation equivariance. This allows to obtain a much more

flexible and universal model, but demands more data to re-learn these biases where they are needed. The ViT is a vivid example, since even 1.3M samples of ImageNet-1k were not enough to outperform CNNs.

However, a one may mitigate the issue from two directions — by an extensive usage of transfer learning and by employing a proper pretraining objective.

Transferring Transformers Unlike RNNs, Transformers demonstrate impressive transfer-learning capabilities. This is being supported by an existing amount of pretrained models for different tasks, languages, domains and input data modalities. But the architecture capabilities go beyond that, and a model trained on a large text corpus can be transferred *between modalities* even if they substantially differ from each other.

The study by Lu at al.[28] demonstrates, that Transformers trained on large text corpora expose universal computation capabilities. It seems that a portion of knowledge, which can be learned from large data, can be universal for all domains and sufficient to achieve a competitive performance with self-attention and FCNN layers being frozen.

2.3.2 Theoretical studies

2.3.2.1 Approximation capabilities

It is not clear at the first glance which class of functions Transformer can approximate, since the model heavily shares parameters along sequence dimension as well as relies on inter-token interaction. However, the work by Yun et al.[29] gives a positive answer on the question: "Are Transformers universal approximators of continuous functions $\mathbf{R}^{L\times d} \mapsto \mathbf{R}^{L\times d}$ on a compact domain?".

▶ **Theorem 2.1.** *Let $t^{h,m,r}$ be a ReLU-activated Transformer encoder block with h attention heads of dimension m and a FCNN hidden dimension r. Let $\mathcal{T}^{h,m,r}$ be a family of functions defined as compositions of $t^{h,m,r}$ blocks. Let $1 \leq p < \infty$ and $\mathrm{d_p}(f_1, f_2) = \left(\int ||f_1(\mathbf{X}) - f_2(\mathbf{X})||_p^p \, d\mathbf{X}\right)^{1/p}$ be a distance between $f_1, f_2 : \mathbf{R}^{d\times n} \mapsto \mathbf{R}^{d\times n}$. Finally, let $\epsilon > 0$.*

1. *For any continuous permutation-equivariant function $f : \mathbf{R}^{d\times n} \mapsto \mathbf{R}^{d\times n}$ there exists a Transformer network $g \in \mathcal{T}^{2,1,4}$ such that $\mathrm{d_p}(f, g) \leq \epsilon$.*

2. *Let $\mathcal{T}_P^{h,m,r}$ be a family of Transformer encoders with positional encodings defined as $\mathcal{T}_P^{h,m,r} = \{g_P = g(\mathbf{X} + \mathbf{E}) \mid g \in \mathcal{T}^{h,m,r} \text{ and } \mathbf{E} \in \mathbf{R}^{d\times n}\}$. Then, for any continuous function $f : D \mapsto \mathbf{R}^{d\times n}$, defined on a compact domain $D \in \mathbf{R}^{d\times n}$, there exists a Transformer network $g_P \in \mathcal{T}_P^{2,1,4}$ such that $\mathrm{d_p}(g_P, f) \leq \epsilon$.*

Proof. For a complete proof refer [29]. The key idea of the proof is that a Transformer can implement a so-called *contextual mapping*, which maps each token to a unique value depending on the whole sequence. ◀

2.3.2.2 Turing-completeness

Another work[30] introduces a proof, that under an assumption of an infinite precision the Transformer is a Turing-complete model without an external memory. The crucial detail there is positional encodings, since without them the model becomes permutation-equivariant and thus unable to recognize even regular languages.

2.3.3 Computational complexity

One of the reasons which led to the Transformer emergence was an inability to parallelize computations of inherently sequential RNNs. Vaswani et al.[7] indeed reported that the proposed architecture was faster than RNNs for input lengths common in practice. However, how really fast are Transformers?

In the following analysis, for simplicity we omit biases and computational costs for vector copying in RNNs. Unless mentioned differently, a plain "complexity" means both time and memory complexity.

Recurrent models Assuming the input token embedding dimensionality d_e and the hidden state of size d_h, the weight matrix of a RNN unit will have a shape $(d_h + d_e) \times d_e$, which leads us to the same complexity.

LSTM decouples a previous model outuput and a hidden state. A forget gate matrix then has the size of $(d_h + d_e) \times d_h$, and it is being followed by an input gate of a doubled complexity. An output gate employs two matrices of shapes $(d_h + d_e) \times d_h$ and $d_h \times d_h$.

By expressing a recurrent model complexity within the $\mathcal{O}$-notation, we obtain a $\mathcal{O}((d_h + d_e) \times d_e)$ complexity for RNNs and $\mathcal{O}((d_h + d_e) \times d_h)$ for LSTM. Since it is a common choice to have $d_e = d = \Theta(d_h)$, for the whole input sequence we obtain identical asymptotic complexities $\mathcal{O}(Ld^2)$ for both models.

Attention-augmented RNNs Let us assume a dot-product attention from the Table 1.1 as the fastest introduced mechanism working with input annotations. Each decoding time step then gets an additional cost of $\Theta(Ld)$, since it is needed to compute a softmax over all the dot products of input annotations with a current hidden state vector. This leads to the $\mathcal{O}(Ld^2)$ complexity for an encoder versus $\mathcal{O}(L^2d^2)$ time and $\mathcal{O}(Ld^2)$ memory complexity for a decoder.

Transformers Transformers eschew recurrence, but employ attention at both encoding and decoding phase. Assuming an input length of L, h attention heads, block input/output dimensionality of d and (for simplicity) a q-k-v dimensionality of d/h, a self-attention complexity may be computed as $3 \times L \times h \times d \times \frac{d}{h} + L^2 \times h \times \frac{d}{h} + L \times (h \times \frac{d}{h}) \times d = \mathcal{O}(Ld^2 + L^2d)$. Same goes for masked-self attention and, under an assumption of an output length $\Theta(L)$, cross-attention.

Position-wise FCNNs have complexities $L \times (d \times d_f + d_f \times d)$. Since most of implementations employ $d_f = 4d$, we estimate a FCNN complexity as $\mathcal{O}(Ld^2)$ and the whole encoder/decoder block complexity as $\mathcal{O}(Ld^2 + L^2d)$.

For the common choices of $d = 768$ and $L = 512$, the dimensionality term dominates and Transformers demonstrate significantly faster computation compared to RNNs. However, attention-based models also demonstrate a quadratic growth of computation time with an input length, while the memory requirements for Transformer grow quadratically as well.

To address the issue, a variety of new Transformer-based architectures were developed. We address the motivation and approaches to processing long sequences with Transformers in the next chapter.

Chapter 3

Efficient Transformers

In this chapter we will overview already existing Transformer architectures maintaining linear complexity. Moving from simpler to more sophisticated models, we will describe different approaches to reducing an attention operation space and time complexity. We will also focus on additional benefits or drawbacks of individual methods which are needed to be taken into account, as well as on appropriate use cases.

As a restrictive limitation, the computational complexity of the original Transformer motivated the community to quest for the solution in order to approximate the architecture with asymptotically faster models. Since the original paper[7] publication, a dizzying amount of so-called *efficient* Transformers emerged[31], each of which introducing its own trade-off between speed and performance.

While the word "efficient" may be interpreted differently, in this work we focus on attention linear time and space complexity. This is motivated by several reasons, but the main one is that comprising all the architectures which accelerate the Transformer architecture is an intimidating amount of work. To give a better view on modern efficient methods, we better focus on strictly linear approaches, which extremely vary and are interesting subjects to compare with each other. They are also interesting from a practical point of view, since they offer the most significant performance boost.

From the definition of efficiency we gave also implicitly follows that we will not cover distillation and parameter sharing techniques in this work. Instead, we focus on ways to improve the key part of the architecture – an attention mechanism – and discuss approaches rather than particular works.

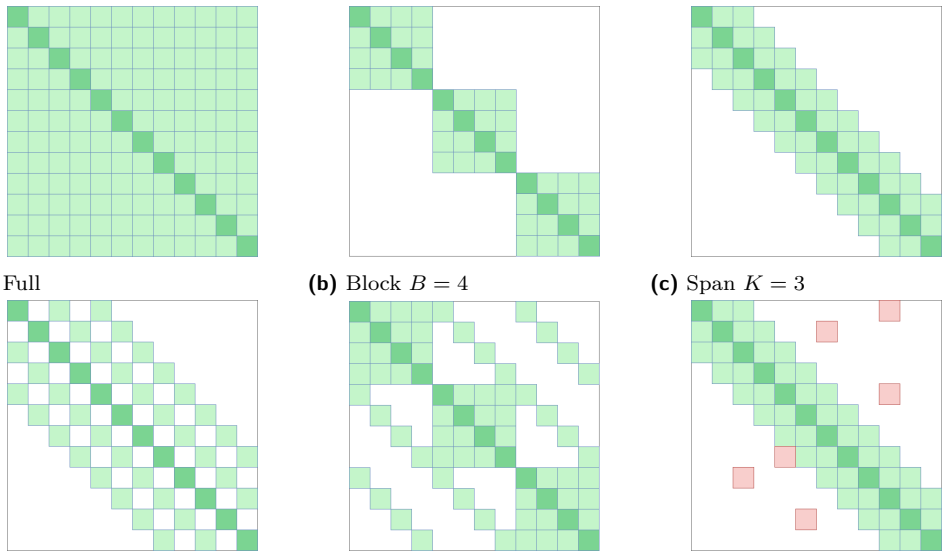
Motivation Most of NLP tasks could be solved via processing shorter segments, e.g. of length 512 or 1024. However, even between problems considered as solved a one can meet tasks which could benefit from a longer context. Problems including document or scientific text generation often place additional constraints on style and terminology consistency, while for text classification tasks a longer input introduces more information to eliminate noise and make a final prediction. Even models of huge capacity, such as GPT-3, can do very little against a fundamental problem of a fixed-length context.

The need for an efficient model becomes even more acute once we go beyond the language processing domain. A more efficient ViT can process larger images without losing in smaller details; audio processing naturally introduces extremely long sequences, which require an extensive pooling before applying a model; the chemical domain also demands an ability to model long protein chains.

A model both able to process very long sequences and having Transformer unbiasedness and flexibility sounds very lucrative, and may open new horizons in data processing. As we will show in this chapter, reducing the asymptotic complexity should not even result in a performance *decrease* — in some settings the model can actually benefit from a reduced complexity, while there exist methods which do not hurt its universal approximation capabilities.

3.1 Sparse attention

Restricting an attention only to a subset of keys is a natural way to improve performance, which was proposed back in the original Transformer paper. Methods described in this section employ some kind of a static pattern, which restricts an attention span from the whole sequence down to some constant-size subset. A variety of approaches can be summarized or decomposed into the following patterns:



(a) Full **(b)** Block $B = 4$ **(c)** Span $K = 3$

(d) Dilated $K = 3$, rate $= 1$ **(e)** Axial, 3×4 input **(f)** Span + random, $r = 6$

Figure 3.1 Sparse attention matrix schemes.

- **Blocks**[32]. Instead of computing an attention across the whole sequence, we can split it first into blocks of size B and compute L/B attentions instead. Each token then attends only tokens from the same block.
- **Constant span**[33][34], or a **sliding window**. Similarly to the previous method, we enforce locality, but instead of block grouping we can restrict an attention span individually for each token. Note that we can limit the span either in an attention matrix (considering only diagonal band of some width) or an input sequence (considering input topology instead of flattened representation), which is actual for two- or more dimensional inputs such as images.
- **Dilated**[33] patterns build gaps between locally attended elements, which allow to capture longer-spanning relationships.
- For tensor inputs an **axial attention** may be applied, allowing only to attend elements along axes (elements strictly above, below, left, right, ...). This method may be especially useful for images or similar kinds of data, which demonstrate correlations along axes.

- A **random attention**[34] may be used in combination with other methods to introduce some global information.
- A fixed amount of **globally attending tokens**[33][34][35] may be used to exchange the information between local blocks/groups. May be either a subset of input tokens or prependable learnable ones, see ETC.

Attending tokens only within some local/constant span is intuitively very restrictive, but if we take a look at the attention matrix of some trained model, we will notice that for a substantial part of tasks a vast majority of attention scores are concentrated in the central band. Models based on static local patterns are able to match or even exceed the base model performance for some tasks, since they additionally introduce a bias towards a local context, which in some settings may lead to a faster and more data-efficient convergence. On the other hand, these methods are inherently inappropriate for tasks where a global attention is necessary, and are significantly inferior to competing models if used on their own.

It is noteworthy that static patterns do not exclude causal masking, thus allowing to build full encoder-decoder architectures. Furthermore, they can be easily combined with other methods to enable them for causal prediction.

Individual patterns are rarely being used on their own; by combining them together we can stack their benefits and eventually obtain pretty powerful models. These methods are also easy to implement, do not require additional optimization on CUDA level (with an exception of complex pattern combinations such as Longformer[33] or ETC[35]) and do not alter parametrization, allowing us to efficiently use the base model weights for transfer learning.

3.2 Memory-based approaches

These approaches leverage additional memory to exchange data globally across an input sequence. A memory can be either in a form of designated input tokens, learnable parameters or special slots to store activations. This approach naturally complements sparse methods and allows to propagate information between local chunks, but is generally incompatible with an autoregressive prediction.

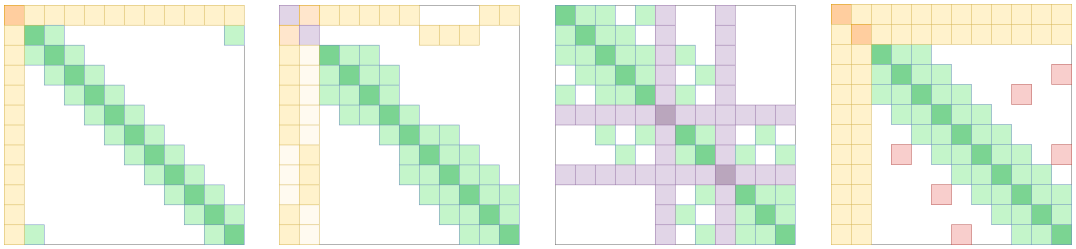
3.2.1 Star Transformer and ETC

Star Transformer The model[36] takes its name from a star-shaped connectivity pattern. This is a direct extension of a sparse sliding-window model, which applies the following attention restrictions:

- A global-attending learnable token (a central vertex).
- A sliding window of range 1 in a *cyclic* manner. It means that an i^{th} query may attend only the global token and keys at positions $\{i-1, i, i+1\} \bmod L$.

A central vertex ("a relay node") is used for a global information exchange, while "satellite nodes" aggregate information from their immediate neighborhood. This architecture introduces the simplest memory-based extension of static pattern models in the form of a single learnable token. The authors also state that the ring topology allows to better propagate information throughout the sequence.

ETC The Extended Transformer Construction[35] architecture leverages a sliding window attention and a larger amount of learnable global tokens. It also proposes relative positional encodings as a way to capture relationships beyond ordering (e.g. *is-a*, *part-of* and other), along with additional masking of certain segments of attention matrix to represent structured inputs. Overall, the model is quite similar to the model above.



(a) Star **(b)** ETC **(c)** Longformer **(d)** BigBird

Figure 3.2 Sparse + memory attention schemes. We employ different color schemes to denote global-to-global, local-to-local and mixed variants of an attention on the ETC figure. The ETC input is masked with certain global-to-local and local-to-local masks to reflect its structure. We also denote with pink the input tokens designated to global attention in Longformer.

3.2.2 Set Transformer

The Set Transformer[37] leverages the Transformer architecture to approximate a set-input function — a permutation-invariant function of input features. It may be proven that all permutation-invariant functions may be represented in the following form:

$$\text{net}(x_1, \ldots, x_L) = \rho(\text{pool}(\{\phi(x_1), \ldots, \phi(x_L)\}))$$

where $\rho(\cdot)$ and $\phi(\cdot)$ are arbitrary continuous functions and $\text{pool}(\cdot)$ is a summation operator. Based on that formulation and an additional notion, that $\phi(\cdot)$ may be a permutation-equivariant function, the authors implement $\phi(\cdot)$ with a Transformer encoder and $\rho(\text{pool}(\cdot))$ as a decoder.

The authors define the following building blocks for a network (we omit LayerNorms for simplicity):

$$\begin{aligned}
\mathbf{MAB}(X, Y) &= H + \text{FCNN}(H) \\
H &= X + \text{MultiHeadAttention}(X, Y, Y) \\
\mathbf{SAB}(X) &= \text{MAB}(X, X) \\
\mathbf{ISAB}_m(X) &= \text{MAB}(X, \text{MAB}(I_m, X)) \\
\mathbf{PMA}_k(X) &= \text{MAB}(S_k, \text{FCNN}(X))
\end{aligned}$$

Multi-Head Attention Block (MAB) is basically a Transformer encoder block with a possibility to perform a cross-attention. ISAB (Induced Set Attention Block) is a way to bypass the quadratic complexity problem by introducing a set of m learnable tokens I_m, and transforming them first with an input sequence before using them to transform an input set. PMA (Pooling by Multi-Head Attention) implements a parameterized pooling operation by transforming a set of k *seed vectors* S_k with a projected input. S_k is somewhat similar to object queries from DETR, and will represent a model output with k typically set to 1.

Combining it all together, we obtain a Set Transformer model which reproduces a functional form above with an encoder of stacked SABs (or ISABs for large sets) and a decoder $\text{Decoder}(X) = \text{FCNN}(\text{SAB}(\text{PMA}_k(X)))$.

3.2.3 Longformer

The Longformer[33] is a yet another model which combines sparse patterns with globally attending tokens:

- First, it exploits a sliding window local attention.

- Additionally, it leverages a dilated sliding window attention with a dilation rate increasing towards last layers. This should allow to efficiently capture longer contexts, since last Transformer layers tend to be more global.
- Finally, the model uses a set of globally attending tokens. Instead of prependable learnable memory, the model uses specially masked input tokens as global. It is important to note that the model uses separate q-k projection weight set for global tokens, doubling the parametrization.

The authors use a base Transformer decoder for seq2seq tasks. Despite their statement of linear complexity in that setting, this is only true under the assumption of an output sequence being much shorter than input (e.g. in summarization tasks). Thus, we will consider this architecture as an encoder-only.

3.2.4 BigBird

The BigBird[34] architecture combines the three previously mentioned approaches — a sliding window, random attention and a learnable memory. More specifically, the model leverages a non-dilated sliding window attention of span $K + r$ randomly attending query-key pairs + g global learnable tokens. The paper defines both the ETC (learnable tokens) and "ITC" (tokens chosen by mask like in Longformer) global attention variants.

The theoretical analysis in this work is of the particular interest, because it demonstrates that a sparse model with a learnable memory is able to preserve the original architecture universal approximation capabilities, as well as Turing-completeness. We will take a closer look at the theoretical findings at the end of the chapter.

3.2.5 Poolingformer

The Poolingformer[38] architecture refines a sliding window attention by constructing an additional processing level. After performing a trivial sliding window attention, an output is again being projected by query, key and value matrices and sent into a sliding window self-attention layer with a larger span. To be able to process very large attention spans, the authors suggest to compress attended keys and values within this span with some pooling function of certain width and stride:

$$\begin{aligned}\text{PooledSelfAttention}(q_i, K, V) &= \text{MultiHead}(q_i, \tilde{K}_i, \tilde{V}_i)\\ \tilde{K}_i &= \text{Pool}(K_{N(i,w)}, S, K)\\ \tilde{V}_i &= \text{Pool}(V_{N(i,w)}, S, K)\end{aligned}$$

where $K_{N(i,w)}$ and $V_{N(i,w)}$ denote key and value matrices for query q_i of span w, $\tilde{K}_i$ and $\tilde{V}_i$ are compressed key-value matrices for the query, S is a pooling stride and K is a pooling window.

Compressing keys and values this way allows to reduce a key matrix S times. Along with mean, max and convolutions, the authors experiment with the other two functions:

$$\text{LDConv}(v_1, \dots, v_K) = \text{softmax}(W\mathbf{v})^T(v_1, \dots, v_K)$$

with $\mathbf{v} = v_{i=\lceil\frac{1+K}{2}\rceil}$ or $\frac{1}{K}\sum_{k=1}^{K} v_k$, $W \in \mathbf{R}^{K\times d}$ and $(v_1, \dots, v_K) \in \mathbf{R}^{K\times d}$.

While the authors report solid results on question answering datasets, it is important to note that the architecture does not allow to compute causal mask, so they used a vanilla Transformer decoder to generate answers. The model performed best when the fraction of pooling-augmented

layers layers was 1/4, and when layers did not share Q, K and V projection matrices between attention levels.

3.2.6 Luna

Linear Unified Nested Attention[39], or Luna, is a recent work on reducing attention computational costs based on introducing an auxilary input sequence of a fixed length. Similarly to the Set Transformer, this sequence acts as a proxy, allowing to factor the large attention operation onto two smaller ones.

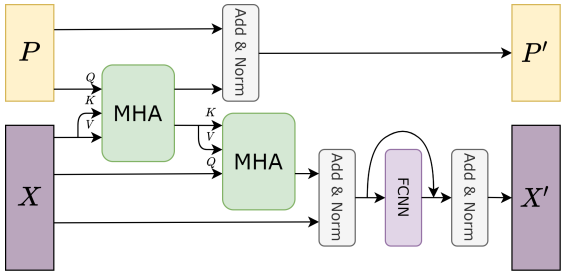


Figure 3.3 Luna encoder block. X denotes an input sequence, P denotes an auxiliary sequence.

Along with an input $X \in \mathbf{R}^{L \times d}$, a Luna encoder block accepts an additional sequence $P \in \mathbf{R}^{n \times d}$. The model first compresses an input sequence by performing a multi-head attention operation ("packing") with $Q = P$ and $K = V = X$. Given the Y_P as a compressed input representation, the model performs then a cross attention between input X and this compressed representation Y_P, "unpacking" an input back:

$$Y_P = \text{MultiHeadAttention}(P, X, X)$$
$$Y_X = \text{MultiHeadAttention}(X, Y_P, Y_P)$$

Analogous to the vanilla Transformer, the layer output is then being given by a token-wise FCNN, applied on summed and normalized X and Y_P. However, an output of the "packing" attention Y_P is then being also summed with P, normalized and sent as an auxiliary output.

Under the assumption that initial P does not contain any information about a decoder input, Luna blocks can be modified for a causal attention implementation. The authors leverage the kernelized attention formulation as in the Subsection 3.5.1, and factor attention matrices between individual attention operations. This allows them to preserve a linear complexity, while also enabling the architecture to be used as a decoder.

3.3 Low-rank approaches

Low-rank methods generally accelerate computation by downsampling or projecting queries and keys to some lower-dimensional space. Unlike kernelized approaches, these algorithms exploit more of linear algebra methods rather than kernel approximation theory, and understand an input as a matrix rather than as a token sequence.

3.3.1 Linformer

The Linformer[40] stands on the assumption of a low-rankness of an attention matrix. The authors refer to the distributional JL lemma and provide some theory to support their intuition, which they embodied into the model, compressing keys and values along the sequence axis.

Before applying an attention, the model linearly projects keys and values of shapes $L \times d_k$ resp. $L \times d_v$ down to $k \times d_k$ resp. $k \times d_v$ matrices. This effectively reduces the complexity to linear, introducing $L \times (d_k + d_v)$ additional parameters. Interestingly, sharing a single matrix between queries, keys and all heads/layers does not result in the performance decrease, allowing to restrain the parametrization growth.

The method resembles both the Memory Compressed Transformer, which applies convolutions along sequence axis, and mixing methods[41] which rely on a concept of information mixing along sequence axis. Due to the information being mixed along the whole axis on projection, a causal mask cannot be applied, limiting the model to be encoder-only.

3.3.2 Nyströmformer

The Nyströmformer[42] is a Transformer architecture, which employs a strategy of an attention matrix approximation based on a subset of (aggregated) queries and keys. Assuming an attention matrix $S = \begin{bmatrix} A_S & B_S \\ F_S & C_S \end{bmatrix}$ split into blocks $A_S \in \mathbf{R}^{m \times m}$, $B_S \in \mathbf{R}^{m \times (L-m)}$, $F_S \in \mathbf{R}^{(L-m) \times m}$ and $C_S \in \mathbf{R}^{(L-m) \times (L-m)}$, a one can reconstruct the C_S matrix according to the quadrature technique from the Nyström method:

$$\hat{S} = \begin{bmatrix} A_S & B_S \\ F_S & F_S A_S^+ B_S \end{bmatrix}$$

where A_S^+ is a Moore-Penrose inverse of A_S. If we choose m as a relatively small constant, the attention computation can be provided in linear time and memory by utilizing the associativity of the matrix multiplication.

A naive application of that method will result in an ability to only approximate attention matrix logits, and will not allow to efficiently approximate the actual attention matrix since we cannot obtain attention scores for F_S without a C_S construction.

The authors propose to flip the operation order, and instead compute the softmax for A_S, F_S and B_S. The intuition behind that is the following: if instead of subsampling queries and keys (called *landmarks*) for A_S we will properly downsample them instead, the relation between a non-landmark query and key may be interpolated with landmarks. Assuming the landmarks $\tilde{Q}, \tilde{K} \in \mathbf{R}^{m \times d}$, we obtain the following attention approximation:

$$\hat{S} = \text{softmax}\left(\frac{Q\tilde{K}^T}{\sqrt{d_q}}\right)\left(\text{softmax}\left(\frac{\tilde{Q}\tilde{K}^T}{\sqrt{d_q}}\right)\right)^+ \text{softmax}\left(\frac{\tilde{Q}K^T}{\sqrt{d_q}}\right)$$

The proposed way to compute landmarks is block-means, which yields satisfying results and is significantly faster than K-means from concurrent works.

Causal masking The causal inference algorithm was not provided in the original work. To prevent a causality break, the model should forbid to attend any keys/key landmarks which were computed using over tokens subsequent to the corresponding query/query landmark. However, we cannot build masks for A_S, B_S and F_S due to the following reasons:

- A simple rejection of "causality breaking" landmarks will remove parts from an attention matrix diagonal.
- Even if we employ another landmarking strategy to preserve the diagonal, the resulting attention matrix will not be lower-triangular but rather a step one, which again would break causality.

Thus, the Nyströmformer is an encoder-only architecture with a $\mathcal{O}(Lm)$ complexity.

3.3.3 Long-Short Transformer

The Transformer-LS[43] is a recent architecture, which decouples long- and short-term dependencies in an input, and leverages local patterns in combination with low-rank projections to process both.

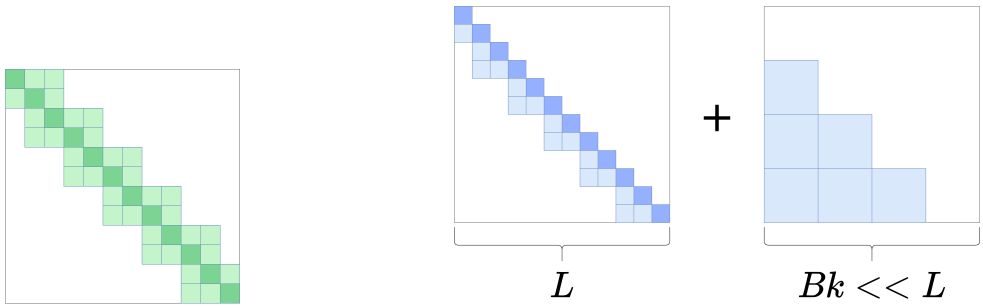
To capture short-term dependencies, the model first applies a fixed-length attention. Unlike the sliding window we mentioned before, the authors divide tokens into groups of size w instead, and allow tokens only to attend within their groups + $w/2$ consecutive tokens to the left and to the right of the block, $2w$ tokens in total. Compared to the sliding window, this pattern allows a faster computation.

Additionally, to capture longer-spanning relationships the authors propose to project keys and values down to a $k \times d$ fixed-length sequence, same as Linformer. However, they condition the mapping on an input, and compute the projection matrix P_i of an i^{th} attention head as:

$$P_i = \text{softmax}(KW_i^P)$$

where the softmax is applied column-wise, K is a key matrix $\mathbf{R}^{L \times d_k}$ and $W_i^P \in \mathbf{R}^{d_k \times k}$ is a learnable matrix.

Instead of applying two mechanisms in two attention layers, the authors simply stack for each query an attended local window and a global low-rank projection of keys and values, obtaining $(2w + k)$ key-value pairs. To avoid the model bias towards the short-term processing (since query-key product magnitudes will be higher within a local window), they additionally apply two LayerNorms before stacking – one for keys and values coming from a local window, and one for global projections.



(a) Non-causal local, $w = 2$ **(b)** Causal local and global

Figure 3.4 Transformer-LS attention illustrated. L stands for an input length, w stands for a local attention span, B stands for blocks count, k stands for block size

Causal masking Causal masking is straightforward for a local attention. For a global attention, the authors suggest first to split an input sequence into blocks, and apply a projection on each block individually. A query q_i then will be only able to attend blocks, which do not contain subsequent keys.

Note that this formulation results in a block-wise triangular matrix, which is strictly under diagonal—for example, for blocks of size b, first $b-1$ queries will not be able to attend any block at all, while the last block will never be attended by any query except the last. This problem bears a resemblance to Nyströmformer causal adaptation efforts. However, the Transformer-LS supports the global attention with local attention windows—even though a global attention matrix will be step-shaped under diagonal, a query token will always have an access to its present and an immediate past.

▶ **Proposition 3.1.** *The Nyströmformer can be enabled for causal attention if we combine its decoder with a sliding window and triangular masks.*

The Transformer-LS complexity is linear both for encoding and decoding, with an additional factor of total projected sequence length k.

3.3.4 H-Transformer-1D

The thought behind this architecture[44] is to model longer-spanning relationships with a gradually decreasing precision. Instead of a low-rank approximation of the whole attention matrix A, we can recursively partition it in the following way:

$$A = \left[\begin{array}{cc|cc} \begin{array}{c|c} A_{11}^{(0)} & A_{12}^{(0)} \\ \hline A_{21}^{(0)} & A_{22}^{(0)} \end{array} & & A_{12}^{(1)} & \\ \hline A_{21}^{(1)} & & \begin{array}{c|c} A_{33}^{(0)} & A_{34}^{(0)} \\ \hline A_{43}^{(0)} & A_{44}^{(0)} \end{array} & \end{array}\right]$$

This partition allows us to make use of a notion, that a numerical rank (a rank of an approximation given some tolerance ϵ) of the off-diagonal blocks the lower the further the block is from the diagonal. Instead of building a low-rank approximation of the whole matrix A, we can keep its central band untouched, and approximate only $A_{12}^{(1)}$ and $A_{21}^{(1)}$ with a higher tolerance and $A_{12}^{(0)}$, $A_{21}^{(0)}$, $A_{34}^{(0)}$ and $A_{43}^{(0)}$ with a lower tolerance.

The proposed algorithm is based on this notion, along with two concepts: coarsening and prolongation. As a part of the algorithm, coarsening consists of averaging pairs of neighboring queries and keys, and summing (as it follows from the derivation in the paper) corresponding pairs of values. By applying this operation l times, we can compute a coarsened attention matrix $A^{(l)}$ which approximates an original attention matrix, and the algorithm uses more coarsened representations for attention matrix blocks located further from the diagonal. Given the attention operation expressed as $D^{-1}AV$, where $D = \mathrm{diag}(A\mathbf{1})$, we can compute its hierarchical approximation as:

$$Y = AV = Y^{(0)} + P^{(0)}(\tilde{Y}^{(1)} + P^{(1)}(\hat{Y}^{(2)} + \dots))$$

where $Y^{(0)}$ is an attention computed within the central bend (special case), $\tilde{Y}^{(l)}$ is an attention computed using super- and under-diagonal blocks of size $N_r \times N_r$ on an attention matrix of l times coarsened input, and $P^{(l)} \in \mathbf{R}^{L/2^{l-1} \times L/2^l}$ is an expansion matrix, which duplicates rows two times. Using this equation, we can similarly compute a normalization matrix D^{-1}, and the final output will be $D^{-1}Y$.

$$\left[\begin{array}{cc|cc|cc} A_{11} & A_{12} & & & & \\ \hline A_{21} & A_{22} & A_{23} & & & \\ \hline & A_{32} & A_{33} & A_{34} & & \\ \hline & & A_{43} & A_{44} & A_{45} & \\ \hline & & & A_{54} & A_{55} & A_{56} \\ \hline & & & & A_{65} & A_{66} \end{array}\right]$$

(a) $A^{(0)}$ central band for $N_r = 2$.

$$\left[\begin{array}{c|c|c|c} & N_r & & \\ \hline N_r & & N_r & \\ \hline & N_r & & N_r \\ \hline & & N_r & \end{array}\right]$$

(b) $A^{(l)}$ coarsened matrix.

$$\left[\begin{array}{c|c|c|c} 1 & 0 & 0 & 0 \\ \hline 1 & 0 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 \\ \hline 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

(c) $P^{(l)}$ expansion matrix 8×4.

Figure 3.5 H-Attention illustration. N_r defines a (diagonal) block size in $A^{(l)}$.

Since the model operates with conventional attention matrices (on coarsened inputs), causal prediction is trivial. The model performs very well on all provided benchmarks, and outperforms concurrent models of significantly larger sice. The result on the LRA ListOps task especially indicates, that a hierarchical inductive bias may be proper for tasks with an explicit hierarchy in input.

3.4 Recurrent approaches

Recurrent approaches take a step back from completely attention-based models, and re-introduce such concepts as a hidden state and sequential processing. Architectures like the Transformer-XL demonstrate, that some form of recurrence may be beneficial and can allow a model to extend its receptive length. However, unlike the seq2seq architecture, Transformer-based recurrent models use recurrence only to supplement an attention mechanism.

3.4.1 Transformer-XL

The Transformer-XL[12] is a generative architecture, which was designed with an aim to increase a Transformer receptive field. One of the long text processing paradigms is to split an input into sections of size W first, and process them independently. This leads to a receptive field bounded by a section length, and an information never propagates between blocks. However, the Transformer-XL overcomes that by storing the last processed section as a hidden state, and conditions the current section keys and values by the previous section output from the previous layer:

$$\begin{aligned} \tilde{h}^{n-1}_{\tau+1} &= [\text{StopGradient}(h^{n-1}_{\tau}), h^{n-1}_{\tau+1}] \\ Q^n_{\tau+1}, K^n_{\tau+1}, V^n_{\tau+1} &= h^{n-1}_{\tau+1} W_q, \tilde{h}^{n-1}_{\tau+1} W_k, \tilde{h}^{n-1}_{\tau+1} W_v \\ h^n_{\tau+1} &= \text{TransformerBlock}(Q^n_{\tau+1}, K^n_{\tau+1}, V^n_{\tau+1}) \end{aligned}$$

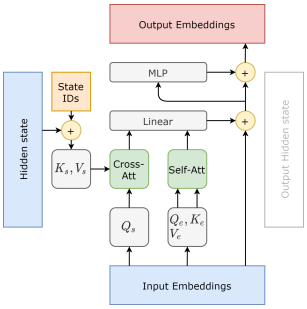
where StopGradient$(\cdot)$ means to stop backpropagation there, $[\cdot, \cdot]$ denotes stacking along the sequence length dimension, τ and $\tau + 1$ are previous and current input sections, h^n_{τ} denotes an output of the n^{th} model layer for the section τ, W_q, W_k and W_v are the n^{th} layer query-key-value projection matrices. To support the recurrence, the architecture overhauls the positional encoding mechanism, and instead of a sum with absolute encodings it dynamically injects relative encodings directly into attention logits computation.

By stacking more layers, we gradually increase the model receptive field, while sectioning allows to achieve a linear complexity w.r.t. to the input length. Additionally, caching allows to **speed up** an inference **up to three orders of magnitude**, since previous activations should not be recomputed. However, the complexity is linear only for very long inputs, and remains $\mathcal{O}(LW^2)$ for a pretty large W (384 resp. 1600 for training resp. inference time).

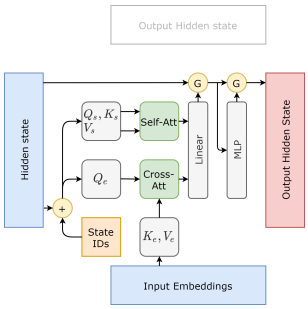
3.4.2 Block-Recurrent Transformer

The Block-Recurrent Transformer[45] augments the Transformer-XL model with a recurrent unit, operating on sequential states and inputs. First, the model partitions a large input into chunks of size N (the authors used 4096), and applies a sliding window causal attention with a window size W, allowing to attend only W previous tokens. The last W chunk outputs are being stored and prepended as the first segment when processing the next chunk, like in the Transformer-XL.

The novelty of the work is a recurrent unit, which is placed instead of a tenth layer in a 12-layer configuration. Unlike the LSTM, the cell operates on sequential hidden states and inputs



(a) Path to generate an output.



(b) Path to generate a new state. G is for gating.

Figure 3.6 Block-Recurrent Transformer recurrent unit. K_e, V_e and K_s, V_s are shared for both paths.

of lengths S and W respectively (the authors suggest $S = W$). The layout of the cell is on the Fig. 3.6, and employs an attention to compute interactions between an input and a hidden state. From the LSTM the model also inherits initialization techniques and training stability issues, while from the Transformer-XL it borrows relational positional encodings.

The additional computational overhead is comparable to an additional attention layer, but the performance gain goes beyond that. The architecture surpasses the already solid Transformer-XL results on generative tasks, outperforming $2\times$ wider configurations. Given a chunk of size N, the architecture complexity remains $\mathcal{O}(NW^2)$, which is linear with respect to an input length.

3.5 Kernelized approaches

Kernelized methods understand an attention matrix as a kernel matrix of a Q-K product. As we already mentioned in the H-Transformer-1D Subsection 3.3.4, we can rewrite an attention as $\text{Att}(Q, K, V) = AD^{-1}V$, where $A = \exp\left(\frac{QK^T}{\sqrt{d_{qk}}}\right)$ and $D = \text{diag}(A\mathbf{1})$.

$$A = \begin{bmatrix} \kappa(q_1^T k_1) & \kappa(q_1^T k_2) & \dots & \kappa(q_1^T k_L) \\ \kappa(q_2^T k_1) & \kappa(q_2^T k_2) & \dots & \kappa(q_2^T k_L) \\ \vdots & \vdots & \ddots & \vdots \\ \kappa(q_L^T k_1) & \kappa(q_L^T k_2) & \dots & \kappa(q_L^T k_L) \end{bmatrix}$$

Figure 3.7 Kernel matrix of an attention operation, assuming that both queries and keys have length L.

More generally, instead of a dot-product exponent we can employ any positive kernel $\kappa(\cdot, \cdot) : \mathbf{R}^{d_q} \times \mathbf{R}^{d_q} \mapsto \mathbf{R}^+$, and rewrite an attention operation for an i^{th} query as[46]:

$$\text{Att}(q_i, K, V) = \sum_{j=1}^{L} \frac{\kappa(q_i, k_j)}{\sum_{j'=1}^{L} \kappa(q_i, k_{j'})} v_j$$

A quadratic complexity is clearly seen there in a form of a double sum over L indices.

If we manage to decompose the kernel back into the dot product of query and key projections $\kappa(q_i, k_j) \approx \phi(q_i)^T \phi(k_j)$, we would be able to rewrite an attention to:

$$\text{Att}(q_i, K, V) \approx \frac{\phi(q_i)^T \sum_{j=1}^{L} \phi(k_j) v_j^T}{\phi(q_i)^T \sum_{j=1}^{L} \phi(k_j)}$$

and achieve a linear complexity.

Particular methods are focusing on different approximations, starting from ordinary activation functions, proceeding with advanced mathematical approximations by Monte-Carlo methods and arriving to parameterized functions.

3.5.1 (Kernelized) Transformers Are RNNs

In the work "Transformers Are RNNs"[47] the authors explore the Transformer in an autoregressive setting and express it as a recurrent network. They employ a simple $\phi(x) = \text{ELU}(x) + 1$ function to approximate a positive kernel, and by combining it with an efficient autoregressive prediction implementation they achieve solid sequence generation results with an up to **4000**× **speedup** compared to the vanilla Transformer.

The RNN formulation is based on the notion that during an autoregressive prediction for a query q_i we compute both factorized kernel attention sums up to index i:

$$\text{MaskedSelfAtt}(q_i, K, V) \approx \frac{\phi(q_i)^T \sum_{j=1}^{i} \phi(k_j) v_j^T}{\phi(q_i)^T \sum_{j=1}^{i} \phi(k_j)} = \frac{\phi(q_i)^T S_i}{\phi(q_i)^T Z_i}$$

To predict the value for a query q_i, we can reuse the sums up to an index $i-1$, and assuming the $\mathcal{O}(1)$ complexity of computing $\phi(\cdot)$ we can achieve a *constant* memory complexity w.r.t. an input length:

$$S_i = S_{i-1} + \phi(k_i) v_i^T$$
$$Z_i = Z_{i-1} + \phi(k_i)$$

These computations resemble the RNN cell state update with the condition of $Z_0 = S_0 = 0$, and are a basis for an autoregressive prediction of all kernelized models. However, the main caveat there is that we cannot efficiently parallelize training in a teacher forcing setting, rendering kernelized autoregressive models notably slow to train without a reconstruction of an attention matrix.

3.5.2 Performer

"Rethinking Attention with Performers"[48] is one of the first works on kernelized transformers with a rigorous theoretical analysis. The authors claim that the following general approximation is able to capture most kernels used in practice:

$$\phi(x) = \frac{h(x)}{\sqrt{m}} [f_1(\omega_1^T x), \ldots, f_1(\omega_m^T x), \ldots, f_l(\omega_1^T x), \ldots, f_l(\omega_m^T x)]$$

where $f_1, \ldots, f_l : \mathbf{R} \mapsto \mathbf{R}$ and $h : \mathbf{R}^d \mapsto \mathbf{R}$ are properly chosen functions and $w_1, \ldots, w_i \overset{iid}{\sim} \mathcal{D} \in \mathcal{P}(\mathbf{R}^d)$ are randomly sampled vectors from a properly chosen distribution $\mathcal{D}$. By plugging in different functions and generating random maps of $\mathbf{R}^{d \times m}$ size, we can obtain approximations of different kernels with different properties. For example, the approximation given by $h(x) = \exp\left(\frac{1}{2}||x||^2\right)$, $l = 2$ and f_1 resp. $f_2 = \sin$ resp. $\cos$ is an unbiased random approximation of a Transformer kernel $\exp(x^T y)$.

The authors explore several possible approximations, and derive conditions under which an approximation will be more robust. For example, the aforementioned variant may potentially

yield negative values for a positive kernel, and thus will potentially result in an abnormal behavior and a numerical instability due to possible almost-zero denominator sums. The resulting approximation should not be unbiased only, but also be positive and have its variance decreasing once an argument approaches zero, which is a common case when attending unrelated tokens.

The main contribution of the work is the **"Fast Attention Via positive Orthogonal Random Features (FAVOR+)"** framework, which can approximate the original Transformer kernel with an arbitrary precision and gives tight variance bounds. The framework is also based on strong theoretical results, which allows to differentiate it from the bigger part of approaches we reviewed before in this chapter.

FAVOR+ Summary The theoretical analysis demonstrates, that the trigonometrical approximator used as an example above has its variance going to infinity once its argument approaches zero. As a usable alternative, they suggest to force a positivity of features and employ a hyperbolic approximation with $h(x) = \frac{1}{\sqrt{2}}$, $l = 2$, $f_1(u)$ resp. $f_2(u) = \exp(u)$ resp. $\exp(-u)$. These improvements result in a strictly lower variance, which approaches zero of an argument tends to zero. Finally, they arrive to the result that forcing a feature map orthogonality yields exponentially lower variance bounds.

Experiments in turn demonstrate, that it is sometimes necessary to redraw a random feature map $W = [\omega_1, \ldots, \omega_m]$ during training, since an approximation error created by some layer propagates further and increases in following layers. Redrawing allows to mitigate the issue, and to obtain impressive results both after training from scratch or after fine-tuning a modified vanilla Transformer.

The work is a confident step towards the theoretic-driven research, and makes a benefit of an outstanding speed of kernelized models. However, while the theoretical outcomes are impressive, practical results may fall behind other approaches, often based on intuition or experimental search.

3.5.3 Learning the kernel

In this work[49] the authors propose another way to approximate a Transformer kernel. They refer to the Bochner's theorem and suggest to learn a distribution, from which random projection maps can be sampled, see Subsection 3.5.2. They employ two approximation strategies for $\phi(x)$— Random Kitchen Sink (RKS) and Positive Random Features (PRF), defined in 3.5.2:

$$\begin{aligned} \Omega &= (w_1, \ldots, w_m) \\ \mathrm{RKS}(x, \Omega) &= \frac{1}{\sqrt{m}}[\cos(w_1^T x), \ldots, \cos(w_m^T x), \sin(w_1^T x), \ldots, \sin(w_m^T, x)] \\ \mathrm{PRF}(x, \Omega) &= \frac{\exp(-||x||^2)}{\sqrt{m}}[\exp(w_1^T x), \ldots, \exp(w_m^T x)] \end{aligned}$$

The main contribution of the work is a study of the methods to approximate a distribution $\mathcal{D}$, from which we sample random feature maps:

- **Gaussian mixtures**. To simplify the method, the authors assume uniform mixture weights and reject the imaginary part of the distribution derived from the Bochner's theorem. The sampled features are then $\Omega = (w_{c,1}, \ldots, w_{C,m})$, with $w_{c,m} = \Sigma_c n_m + \mu_c,\ n_m \sim \mathcal{N}(0, I)$ and with learnable covariance matrices Σ_c and means μ_c of individual components. The PRF method may be additionally sped up by restricting covariance matrices to be diagonal.
- **Fast-Food**[50]. To speed up an Ωx product computation for large m, the authors suggest to approximate Ω with the product of Hadamard matrices: $\Omega \approx \frac{1}{\sigma\sqrt{d_q}} SHG\Pi HB$. Here $\Pi \in \{0, 1\}^{d_q \times d_q}$ is a permutation matrix, $G = \mathrm{diag}(d_g),\ d_g \sim \mathcal{N}(0, I_{d_q})$, H is a Hadamard

matrix, B is a random diagonal $\{+1, -1\}^{d_q}$ matrix, and S is a random diagonal scaling matrix to force the non-uniformity of row vector lengths. Such a computation is log-linear in terms of d_q. Since the resulting matrix is $d_q \times d_q$, the authors of the original Fast-Food method propose to stack several of them, considering m padded to the proper size.

To make the whole transform learnable, the authors propose to "unfreeze" S and optionally G and B. The experiments were conducted with all the three matrices learnable.

- **Generative models**. The authors adapt a deep generative model (DGM), which maps a noise sample from a prior distribution $(\eta_1, \ldots, \eta_m) \sim \mathcal{D}(\mathbf{R}^{d_q})$ to a random feature map $\Omega = (w_1, \ldots, w_m)$. The experiments were conducted using the $\mathcal{D} = \mathcal{N}(0, 1)$ and a four-layer LeakyReLU-activated network + a tanh-activated output layer as a DGM.

Combining the two approximation and three sampling strategies, the authors evaluate six models in total.

The RKS models show solid results on the LRA benchmark, while the PRF-based variants outperform all the baseline models. The six tested architectures also perform on par with the vanilla architecture on the GLUE benchmark, indicating that the method does not hurt the model performance on shorter inputs. Additionally, the authors provide a proof that their model remains Turing-complete.

3.6 Analysis

It is not an easy task to compare a performance of all introduced models. The main reason is that there is a lack of benchmarks which can properly estimate how these models will perform "in a battle". The most common benchmark for efficient models is the Long Range Arena[51], which was specifically designed to test a model inductive bias rather than a pretraining strategy. On the other hand, the tasks in the dataset are half-artificial and do not really correspond to practical Transformer use cases.

Instead of a direct comparison of individual architectures, we will discuss their inductive biases and proper use cases instead. We suppose that a one willing to employ an efficient architecture will conduct multiple experiments anyways, and instead of giving a concrete (and probably wrong) answer on the question "which is best" we will try to narrow the choice.

Computational complexity All the introduced models are linear in terms of memory consumption or computing time, but additional factors introduced are extremely varying and may render particular models impractical in some settings. We summarize our findings regarding the computational complexity in the Table 3.1.

The **sparse** models demonstrate fast computation if their attended query-key pairs are organized block-wise. The more an attention pattern diverges from a block-wise form (which can be easier rearranged into a dense matrix multiplication), the slower the operation goes, up to the moment when an additional optimization at the CUDA level becomes essential (e.g. Linformer dilated patterns).

The **low-rank** methods demonstrate a more complex computation FLOP-wise, but this is being mitigated by an inherently dense computation along an overall better performance, resulting in a slightly better performance-accuracy tradeoff[43].

The **kernelized** methods in turn demonstrate a mediocre performance score-wise, but belong to the fastest methods for causal inference thanks to their cumulative sum sequential computation. However, the cost may be unbearable — these architectures are exceedingly slow during training. The the sequential inference should be conducted during training as well, disallowing us to benefit from a parallelized teacher forcing. A possible solution could be to reconstruct an attention matrix back, but this reintroduces again a quadratic memory complexity.

Finally, the introduced **recurrent** methods demonstrate a significant speedup for autoregressive tasks. However, the benefit may be primarily seen only at sequence lengths going far beyond the standard 512 tokens, while remaining comparable with the vanilla model for shorter inputs.

We denote architecture-specific parameters in the following way: B stands for an attention block size, K, K_1 and K_2 for sliding window attention spans, d_i for individual input tensor dimensions, r for random query-key attention pairs amount, g for global tokens amount, P for the Luna memory sequence length, k for the Linformer projection dimensionality, m for the Nyströmformer landmarks count, N_r for the Hierarchical Transformer block numerical rank, W for a segment size in the recurrence-augmented models, M for Performer feature map dimensionality.

Approximation capabilities An important question is: what do we sacrifice when employing an approximated attention mechanism? We already know that the vanilla Transformer is a universal approximator, but will it be if we replace an attention mechanism by a sparse pattern? Or a low-rank projection? Or any other method?

There is no exhaustive answer to these questions yet; however, a recent progress on the field includes a work by Yun et al.[52], which extends their findings about the original architecture to sparse attention methods. The surprising result is that under some (pretty intuitive) conditions a sparse attention model with $\mathcal{O}(L)$ complex connectivity pattern can retain the original model universal approximation capability.

▶ **Theorem 3.2. (informal)** *A sparse Transformer remains a universal approximator if it satisfies the following conditions:*

1. *Every token can attend to itself.*
2. *There exists a "chain" which connects all tokens.*
3. *Each token is reachable from an each token within N "hops", where N is the model layer count.*

Proof. For a proof see[52]. ◀

All the three conditions are very intuitive, and the main contribution of the work is to confirm and support the assumption that information needs to propagate between all the input tokens in some way—at least between layers. However, there is no free lunch, and a sparsified model tends to need more layers than the original Transformer to become a universal approximator.

We have not found yet any works explicitly proving a universal approximability of low-rank or kernelized models; however, we can speculate a bit about the latter. The original model is a universal approximator, and the kernelized methods such as Performer or aforementioned learnable variants tend to directly approximate it; we suppose that these methods can also expose a universal approximability property:

- The Performer is claimed to approximate the softmax kernel with an arbitrary precision, which may imply an arbitrary approximation precision for the whole model, although with significantly worse bounds due to error propagation.
- GMM and DGN models are universal approximators of distributions. This may allow them to approximate the kernel spectral distribution with an arbitrary precision, thus the Transformer kernel, thus the whole model.

▶ **Conjecture 3.3.** *The Performer, PRF-GMM and Generative-PRF can be proven to be universal approximators.*

Type	Model	Time Complexity	Memory Complexity	Causal?	Notes
Sparse	Blocks	$\mathcal{O}(LB)$		Yes	Can be easily combined together or with other methods.
	Span	$\mathcal{O}(LK)$		Yes	
	Dilated	$\mathcal{O}(LK)$		Yes	
	Axial	$\mathcal{O}(L\sum_i d_i)$		Yes	
	Random	$\mathcal{O}(r) + X$		Yes	
Memory	Global tokens	$\mathcal{O}(Lg) + X$		No	
	Star/ETC	$\mathcal{O}(L(K+g) + g^2)$		No	
	Set	$\mathcal{O}(L(m+k))$		No	
	Longformer	$\mathcal{O}(L(K+g) + g^2)$		No	
	BigBird	$\mathcal{O}(L(K+g) + g^2 + r)$		No	
	Poolingformer	$\mathcal{O}(L(K_1 + K_2))$		No	
	Luna	$\mathcal{O}(LP)$		Yes	
Low-Rank	Linformer	$\mathcal{O}(Lk)$		No	
	Nyströmformer	$\mathcal{O}(Lm + m^3)$		No	
	Transformer-LS	$\mathcal{O}(L(k+K))$		Yes	
	H-Transformer	$\mathcal{O}(LN_r)$		Yes	
Recurrence	Transformer-XL	$\mathcal{O}(LW^2)$	$\mathcal{O}(W^2)$	Only	Linear only for very long inputs
	Block-Recurrent	$\mathcal{O}(LW^2)$	$\mathcal{O}(W^2)$	Only	
Kernelized	Linear	$\mathcal{O}(L)$		Yes	Slow autoregressive training, $\mathcal{O}(1)$ causal mem. comp.
	Performer	$\mathcal{O}(LM)$		Yes	
	Learnable	- (linear w.r.t. L)		Yes	

Table 3.1 Computational complexities of the reviewed attention mechanisms. The "-" instead of the learnable kernel complexity indicate that there are multiple models with different complexities.

Use cases Here we denote possible use cases of individual architectures, based on their inductive biases and performance on different tasks:

The **sparse attention** methods work best as an additional attention method. They may enable the algorithm for a causal prediction, as well as introduce an explicit local attention for a long-range attention approximation. They can also be worth to try for tasks with a low amount of data provided, since these models consider a much lesser amount of inter-token relationships, and should be easier to generalize.

The **low-rank** methods can be appropriate for tasks exposing longer-spanning relationships with a low intrinsic dimensionality. Such tasks can be classification, text matching or any other problems implicitly involving sparse input features (such as class-specific words in input).

The **kernelized** approaches are very similar to the low-rank ones, since they also create a compressed key-value representation. Along that, they additionally offer a computationally efficient framework for causal prediction, which enables the model to output very long sequences with minimal costs. We suppose that a proper use case would be an integration into an already deployed model as a drop-in replacement, since these methods are mostly built around an approximation of the vanilla model attention matrix. Another potentially suitable tasks for this class of models are sequence-to-sequence involving a very long sequence output; however, to become applicable these methods should be additionally combined with implementation tricks allowing to overcome training difficulties.

In this chapter, we have reviewed the bigger part of the linear attention architectures. Now, it is time to introduce an approach by ours, which also retains a linear complexity while being able of causal prediction.

Chapter 4

Proposed Variants

In this chapter we propose our variants of a linear attention mechanism. We will introduce our approaches, explain a motivation, principles and limitations behind them. Then we will discuss experimental setup, choose datasets and set up a pipeline for evaluating models in different settings. We will show that our proposed models achieve a competitive performance compared to a base model and other existing efficient approaches, while maintaining linear complexity both in time and memory.

The main contribution of this work is a development and evaluation of some *new* methods. Both described architectures are kernelized approaches and were largely motivated by the Performer work, thus sharing their advantages and disadvantages with other models of the same category.

4.1 Feedforward Kernel

The intuition standing behind the first model is the following: if there exists a function $\phi(\cdot)$, which corresponds to a projection function of a softmax kernel $\kappa(q_i, k_j) = \exp(q_i^T k_j) = \phi(q_i)^T \phi(k_j)$, then it can be approximated via a universal approximator. By choosing a proper architecture to approximate $\phi(\cdot)$, we can achieve a precise approximation of a softmax function. Furthermore, since $\phi(\cdot)$ is learnable, the resulting approximation can be even more suitable for a given task.

We explore several choices of these architectures, along with some additional ways to increase their performance and stabilize convergence. However, since we have chosen LRA[51] as a primary testing benchmark, we have an additional constraint of no more than 10% of additional parametrization. This is a substantial limitation, but it allows to shift an attention more to the model inductive bias.

Let us recall the kernelized model formulation:

$$\mathrm{Att}(q_i, K, V) = \sum_{j=1}^{L} \frac{\kappa(q_i, k_j)}{\sum_{j'=1}^{L} \kappa(q_i, k_{j'})} v_j \approx \frac{\phi(q_i)^T \sum_{j=1}^{L} \phi(k_j) v_j^T}{\phi(q_i)^T \sum_{j=1}^{L} \phi(k_j)}$$

The $\phi(\cdot)$ in the equation above is the target function we would like to approximate. Along with a reduced parametrization, additional contraints put on the approximator would be:

- A $\mathcal{O}(1)$ computational complexity for a sequence element.
- A relatively small multiplicative constant under the O-notation.

Even though GPUs are optimized for dense neural networks (which are a foundation of the proposed method), we still want the FCNN to be of reasonable size, since even a constant complexity does not guarantee a fast computation.

Another important subject to consider is a final activation function. The authors of the Performer (see Subsection 3.5.2) in their work noted, that an approximation of $\phi(\cdot)$ potentially yielding zero or negative values may result in an abnormal behavior. Indeed, even without a rigorous mathematical analysis it may be seen, that this setup may eventually end in a denominator close to zero, destabilizing the whole network and leading to meaningless outputs (or even NaNs). So, it is important to limit the approximator output to positive values, and preliminary experiments supported that.

N.B.: we name our method "Feedforward Kernel" to avoid a confusion with a learnable kernel we reviewed in the previous chapter. However, we have experimented with neural networks going beyond simple FCNNs.

4.1.1 Architectures

We start with the simplest neural network:

$$\phi(X) = \text{softplus}(XW)$$

where W is a learnable matrix. A one may use linear layers which include biases, but we prefer to disable them since they negatively interact with a regularization introduced later, and disabling them caused no negative effect during preliminary experiments. The softplus function was arbitrarily chosen to force a positive output; we suppose that using $\text{ELU}(\cdot) + 1$ or any other "soft" ReLU approximation instead would result in a similar performance.

Following the[53], we also experiment with orthogonal initialization and regularization as ways to improve convergence, and to avoid possible negative effects due to parameter sharing along sequence or layer stacking. The regularization we employ is very simple:

$$\text{Reg}(W) = \lambda||WW^T - I||_2^2$$

We can further extend the neural network by stacking more modules to obtain a potentially more powerful approximator:

$$\phi(X) = \text{softplus}(\text{FCNN(X)}W)$$

As an alternative choice, we have also experimented with Gated Linear Unit networks:

$$\text{GLU}(X) = XW_t \odot \sigma(XW_g)$$
$$\text{GLU}_{\text{out}}(X) = \text{softplus}(XW_t) \odot \sigma(XW_g)$$

where W_t and W_g are learnable matrices, σ is a sigmoid function, and $\odot$ represents an element-wise product. This linear layer has a doubled parametrization, but offers an activation function which is individual per input element and is conditioned on input elements, allowing to represent more complex functions. We force an orthogonality of this layer too by regularizing the transform matrix W_t. We refer this unit as OGLU.

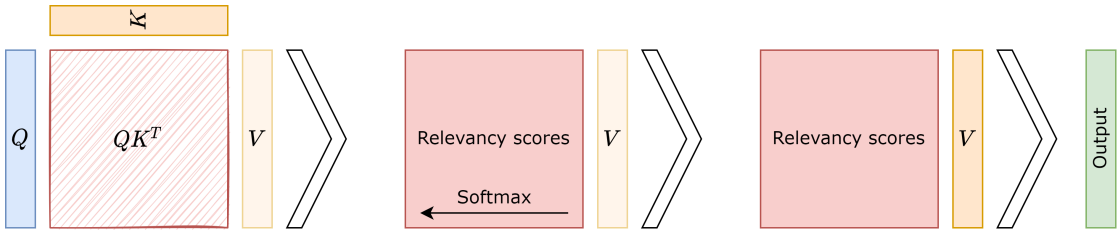
To mitigate the parametrization growth, we can assume that gating requires less information than transforms. Thus, we can approximate the $W_g \in \mathbf{R}^{d \times d}$ with, say, two low-rank matrices of sizes $d \times r$ resp. $r \times d$, where $r < \frac{d}{2}$ is the W_g approximation rank. We refer this unit as A(pproximated)OGLU.

Finally, we hypothesized that it may be beneficial to model query-key projection distribution instead of a mapping, and tried a variational autoencoder as a non-deterministic approximation of $\phi(\cdot)$. In this setting the model is being trained to restore its inputs, while the projection for a particular query or key is given by a vector sampled from a conditioned latent distribution.

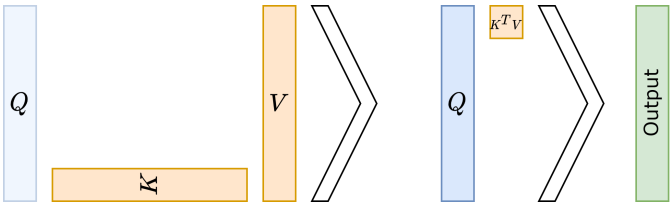
For any models employed, it is beneficial to apply them individually per attention head. The reason is that their additional parametrization and computational costs grow quadratically with a hidden dimension; by splitting it into h heads, we obtain a $\mathcal{O}((\frac{d}{h})^2 h) = \mathcal{O}(\frac{d^2}{h})$ reduced complexity.

4.2 SimpleTRON

The SimpleTRON stands for "Simple Transformer with $\mathcal{O}(N)$ complexity". This is the model which was used as a baseline during preliminary experiments with other models, but promising results attracted our attention and eventually led to the deeper research of the architecture.



(a) Transformer



(b) SimpleTRON

Figure 4.1 Transformer and SimpleTRON attention pipelines. Rejecting the softmax allows to use the matrix associativity to compute a compressed input representation to update queries. Matrices about to perform a product are emphasized.

The SimpleTRON eschews *any* nonlinearity applied on attention matrix logits. It does not approximate a softmax or any similar function; instead, it relies on information mixing along sequence dimension via contractive dot products. Using the kernelized Transformer formulation, the model may be expressed as:

$$\text{SimpleAtt}(q_i, K, V) = \frac{\phi(q_i)^T \sum_{j=1}^{L} \phi(k_j) v_j^T}{\cancel{\phi(q_i)^T \sum_{j=1}^{L} \phi(k_j)}}, \text{ where } \phi(\cdot) \text{ is an identity}$$

or simply in its matrix form (we omit biases for simplicity):

$$\text{SimpleAtt}(Q, K, V) = \frac{1}{\sqrt{L}} Q(K^T V) = \frac{1}{\sqrt{L}} X W_Q W_K^T X^T X W_V$$

where W_Q, W_K and W_V are query, key and value projection matrices, and the $1/\sqrt{L}$ normalization term is a constant we have found helping to achieve a more stable convergence. While

$W_Q W_K^T$ may be replaced with a single matrix, we keep the original Q, K and V framework due to the reason described in the experiments section.

Since the model is kernelized, this allows us to make use of the recurrent autoregressive prediction formulation:

$$\begin{aligned} \text{MaskedSimpleAtt}(q_i, K, V) &= q_i^T S_i \\ S_i &= S_{i-1} + k_i v_i^T \\ S_0 &= 0 \end{aligned}$$

While this method may not seem intuitive or correct at first sight, for some tasks this may be a beneficial inductive bias.

Reduced parametrization Additional experiments were conducted on possible block rearrangements—for example, we observed that an attention layer is a linear map, and that value vectors undergo three linear transforms before entering a block FCNN—value projection, attention and a projection after heads concatenation. We removed the last linear transform in a self-attention layer, and observed a notable performance and convergence stability improvements.

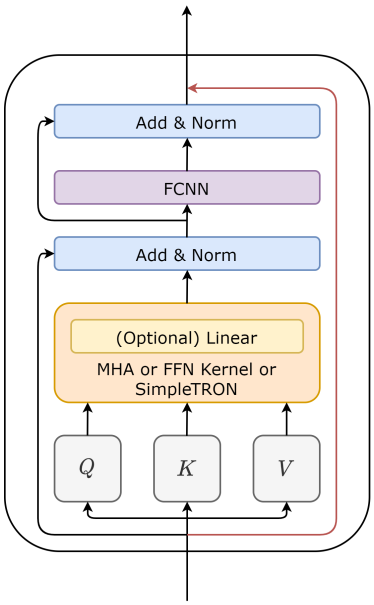


Figure 4.2 Scheme of the encoder block we are considering on this work. The additional skip connection is denoted with red.

Skip connections We also experiment with an additional skip connection over the whole block, just before the last LayerNorm. Since we perform a dot product along the sequence dim, attention outputs just before a FCNN may have a high variance, and an additional skip connection should stabilize the model.

4.3 Experiments

4.3.1 Setup

4.3.1.1 Dependencies and environment

All the models were implemented in PyTorch 1.11.0. Compared to Tensorflow, PyTorch does not require building the whole computation graph before execution, which allows us to develop and test models significantly faster. Another benefit of PyTorch is an ability to modify existing models on-the-fly, by simply plugging in different modules into an existing model before execution.

Due to the different availability of execution environments, experiments were conducted pretty heterogeneously:

- Major fraction of experiments was executed on Google Colab, in particular in Nvidia V100 and A100 environments.
- A fraction of experiments (primarily AG News setups of SimpleTRON) was executed on the private RTX3090-equipped machine.

Since none of implemented models got BatchNormalization layers, we suppose that an effect of different runtimes will be negligible. This also allows to leverage the gradient accumulation technique to simulate larger batch sizes.

Additionally, we use the Huggingface library as a source of reference models for setups other than LRA. The library provides both PyTorch and Tensorflow implementations of majority of Transformers, as well as execution-ready tokenizers for each.

4.3.1.2 Datasets

Long Range Arena[51] We have chosen LRA as a primary testing benchmark. These tasks were specifically designed to test an inductive bias by training a model from scratch without sophisticated pretraining schemes, which could obscur the true model performance. The benchmark comprises the four primary tasks:

- **BPE-encoded text classification**. This task is a binary classification of IMDB review texts, encoded as byte pairs to produce long sequences. The task is more challenging than a word-level classification, since byte pairs offer less information and it is needed to re-learn interactions between them. Following the benchmark rules, we choose $4K$ as a sequence length.
- **BPE-encoded text matching**. The task is a binary classification, whether two BPE-encoded input texts from the ACL Anthology Network have a citation link. Each text is being encoded separately, and their concatenated representation is then being passed to an output layer. We again choose $4K$ as a sequence length, according to the LRA challenge setup.
- **ListOps**. The dataset is composed of nested digit sequences, each of which is coupled with a reduction operation (max, mean, median, ...). The task is a ten-class classification of a final result prediction. We choose $2K$ as a seuquence length used to obtain leaderboard results.
- **Pathfinder**. The task is to predict, whether two endpoints on a 32×32 image are connected by a dashed line. There are also many false lines on an image, which lead to nowhere and make the task challenging for convolutional networks too. An image is being fed per-pixel, resulting in a sequence of length 1024.
- **Pixel-level image classification**. The task is a CIFAR-10 image classification, but with inputs fed into model as a sequence. A flattened 32×32 image results in a sequence length of 1024.

AG News AG News is a subset of AG's corpus of news and articles from the four largest categories—"World", "Sports", "Business" and "Sci/Tech". Tokenized sequences mostly follow conventional Transformer input lengths (up to 512), so we use this dataset to test architectures in a usual text processing pipeline setting.

Due to the large number of tested models and limited computation sources, we restrict ourselves only to ListOps, text classification and retrieval LRA tasks. Additionally, we select particular architectures based on the performance on these tasks and test them on AG News.

4.3.2 Results

4.3.2.1 LRA

For the LRA benchmark, we have implemented our models to be as close to the original Jax implementations as possible, in terms of a layer arrangement and applied regularizations. We also use the same hyperparameters such as block count, query-key-value and hidden dimensionalities, head count, optimizer, learning schedule and other. The only exception we did was the β_2 set to 0.999, since with $\beta_2 = 0.98$ we observed significantly worse results and convergence instabilities.

Feedforward kernel While the performance on the ListOps is a bit underwhelming, the model performs competitively on the classification task, and beats the baseline models by 5% score on the matching. The FCNN realizing the kernel is composed of linear/GLU layers, which use a head dimension as their hidden width (i.e. no expansion or compression).

In the Table 4.1, we denote the architecture-dependent complexity multiplier as C. According on the aforementioned configuration, we can expand C to $\mathcal{O}(ld^2)$ for linear/GLU units or $\mathcal{O}(ldr)$ for AGLU, where l is the layer count. We use $r = \frac{d}{4}$, which leads to a 25% parametrization decrease compared to the original GLU.

The preliminary experiments were conducted on the classification task, and according to the results on it we have chosen an orthogonal regularization coefficient as $\lambda = 0.1$. However, it seems to cause a significant performance hit on the matching task, cutting off roughly 1% of scores. We suppose that the parameter is highly task-specific, and is easy to set into an over-regularization.

Finally, we point out to the fact that the VAE kernel did not exceed the random prediction performance at any setup (layer count and hidden dimensions).

SimpleTRON The SimpleTRON has beaten all the baseline architectures in all three tasks. However, the convergence could not be achieved with the basic architecture — either there should be a skip connection or should not be a linear layer after heads concatenation.

We denote model configurations with an additional skip connection as "-Res", while variants with a linear layer after head concatenation as "-L".

4.3.2.2 AG News

On the AG News dataset, we conducted experiments with the SimpleTRON model along with some chosen Feedforward kernel variants. We employ the training setup from [54], and reimplement our model by replicating it and the Huggingface BERT-base[55] configuration.

In contrary to the LRA results, we have found our models underperforming in this setup. Both the Feedforward kernel and the SimpleTRON achieve $1 - 1.5\%$ less scores than the BERT reimplementation. Interestingly, in the from-scratch settings the SimpleTRON performs on par with the original model — we suggest that the weight incompatibility between architectures, while not being the only explanation, plays the role here.

Model	Complexity	Classif.	Matching	ListOps
Random	$\mathcal{O}(1)$	50.00	50.00	10.00
Transformer	$\mathcal{O}(L^2)$	64.27	57.46	36.37
Synthesizer	$\mathcal{O}(L^2)$	61.68	54.67	36.99
Sinkhorn Trans.	$\mathcal{O}(B^2+(N/B)^2)$	61.20	53.83	33.67
Sparse Trans.	$\mathcal{O}(L\sqrt{L})$	63.58	59.59	17.07
Reformer	$\mathcal{O}(L \log L)$	56.10	53.40	37.27
Local Attention	$\mathcal{O}(LK)$	52.98	53.39	15.82
Longformer	$\mathcal{O}(LK)$	62.85	56.89	35.63
Linformer	$\mathcal{O}(L)$	53.94	52.27	35.70
BigBird	$\mathcal{O}(L)$	64.02	59.29	36.05
Linear ELU	$\mathcal{O}(L)$	65.90	53.09	16.13
Performer	$\mathcal{O}(L)$	65.40	53.82	18.01
GMM-RKS	$\mathcal{O}(L)$	66.20	58.74	18.15
FastFood-RKS	$\mathcal{O}(L)$	65.91	57.47	18.20
Generative-RKS	$\mathcal{O}(L)$	66.37	59.02	17.80
GMM-PRF	$\mathcal{O}(L)$	62.70	59.64	36.95
FastFood-PRF	$\mathcal{O}(L)$	64.69	67.90	37.25
Generative-PRF	$\mathcal{O}(L)$	62.39	67.18	37.10
Linear kernel	$\mathcal{O}(CL)$	65.77	73.51	18.54
1× GLU	$\mathcal{O}(CL)$	65.82	72.17	18.67
2× GLU	$\mathcal{O}(CL)$	65.99	73.36	18.42
3× GLU	$\mathcal{O}(CL)$	65.87	72.60	18.68
Orth. linear kernel	$\mathcal{O}(CL)$	65.86	72.63	18.19
1× OGLU	$\mathcal{O}(CL)$	65.95	72.50	18.45
2× OGLU	$\mathcal{O}(CL)$	66.02	72.96	18.32
3× AOGLU	$\mathcal{O}(CL)$	66.06	72.57	18.45
Variational	$\mathcal{O}(CL)$	FAIL	FAIL	FAIL
Simple	$\mathcal{O}(L)$	66.75	73.92	37.45
Simple-L	$\mathcal{O}(L)$	FAIL	FAIL	FAIL
Simple-Res	$\mathcal{O}(L)$	66.65	**74.83**	37.10
Simple-Res-L	$\mathcal{O}(L)$	**66.71**	73.59	**37.55**

Table 4.1 Baseline and proposed models on the three LRA tasks. We denote sequence length as L, attention span as K and Sinkhorn model block size as B. C is an architecture-dependent multiplier explained in 4.3.2.1.

Model	BERT[54]	BERT (reimpl.)	FFK Lin.	FFK. 2× GLU	Simple
Accuracy	95.2	94.2	93.0		92.7

Table 4.2 AG News fine-tuning results.

Blocks	BERT	*SimpleTRON*
1	89.12	90.90
2	90.38	90.32
3	90.28	90.44
4	90.66	90.41
5	90.11	90.35
6	90.30	90.22
7	90.68	90.19
8	90.68	89.98
9	91.13	89.95
10	91.21	89.94
11	90.78	89.43
12	90.10	89.90
6*	92.70	92.30
12*	94.20	92.70

Table 4.3 SimpleTRON training from-scratch results on AG News, with respect to the number of layers of the model. We denote fine-tuning results with an asterisk.

4.3.3 Discussion

Computational complexity By swapping the q-k-v product matrices and avoiding any kind of approximation we have reached a truly linear complexity with a respect to the input length. The most of linear attention approximations are in fact omitting high architecture-dependent multiplier, which should be taken into account in practice; however, the SimpleTRON does not have any intermediate steps in the q-k-v product, which places it among the fastest models.

On the other hand, the most of efficient architectures are considering $L >> d$. For the common choice of an input length as 512 the dimensionality term dominates in the attention matrix computation. This results in the $L \times L$ attention matrix being significantly smaller and faster to compute than the $d \times d$ intermediate result in the kernelized model, which makes these models only feasible at input lengths longer than some lower bound larger than L.

Normalization The kernelized attention formulation $D^{-1}AV$ requires a normalization matrix D^{-1} to be computed. Since we have forced a positive output of the Feedforward kernel, we can compute it in the ordinary way, by using the factorized formulation we recalled in the Subsection 4.1.

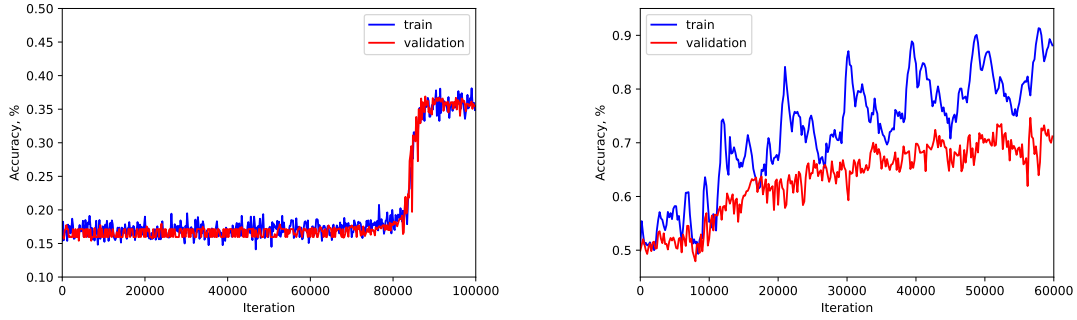
On the other hand, we have completely rejected normalization in the SimpleTRON. Computing a normalization matrix as in 4.1 results in near-zero values and invalid outputs. We have found the model giving satisfying results during preliminary experiments on the CIFAR-10 and LRA benchmark without it; we got no explanation for this. We can suppose that an intermediate result computed by the SimpleTRON got very little in common with the softmax approximation made by the Performer or RKS/PRF learnable models.

We additionally apply the $1/\sqrt{L}$ normalization term to keep the k-v intermediate output variance at the same magnitude. We have found it stabilizing the convergence.

Convergence speed Compared to the vanilla Transformer, the Feedforward kernel converges slightly faster. The SimpleTRON demonstrates an opposite, requiring $\times 1.5 - 3$ steps to converge.

On the AG News both model categories converged significantly faster than the vanilla model, but with significantly worse results.

The SimpleTRON with a linear layer demonstrated a significant convergence instability compared to the other variants. In the AG News setting it may take several attempts and reinitializations to begin the training.



(a) ListOps **(b)** Matching

Figure 4.3 SimpleTRON example training plots.

Attention matrix reconstruction Both kinds of architectures did not manage to reconstruct the original attention diagonal pattern. Since we did not approximate the softmax directly (like the Performer) or indirectly by applying a certain approximation strategy (like the learnable RKS/PRF kernels), it is possible to learn a completely different pattern.

(a) 3× AGLU kernel. **(b)** SimpleTRON

Figure 4.4 Example reconstructions of attention matrices for a 4001×4001 input of the BPE classification task. The value range is approximately $[0 - 0.005)$ for the left matrix and $(-150; 150)$ for the right. The right matrix was extracted as unnormalized (without the $1/\sqrt{L}$ term).

Backward compatibility An additional experiment was conducted with the SimpleTRON — fine-tuning the vanilla architecture with the weights obtained from the Simple Attention model. Within this setup, the model has shown an interesting behaviour: having about 30% less

trainable parameters and in fact no ability to learn pairwise relation between the tokens in the sequences, the model trained up to the original accuracy of the vanilla Transformer, in much less number of training epochs. The result is quite surprising, given the difference of intrinsic representations between both architectures.

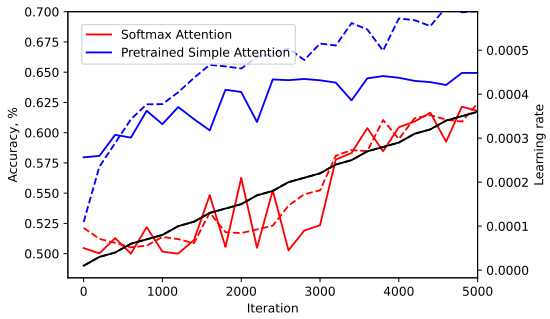
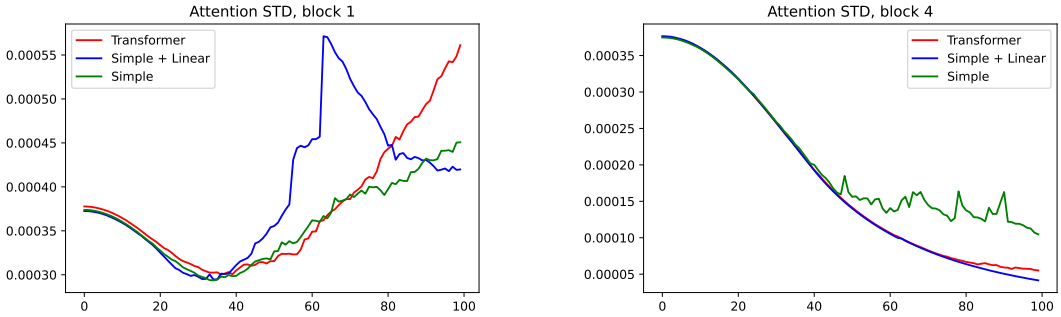


Figure 4.5 Training plot of the original Transformer fine-tuning on the LRA text classification task. Initial weights are given by the trained SimpleTRON model.

Weight symmetry Results on the AG News demonstrate, that a successful training is possible only in the case when the number of blocks is low (i.e. 4 blocks for the LRA text classification task). Even though, at the very early stage of training deeper models with a base Simple attention (with a linear layer and no skips) are on-par with the vanilla Transformer, and after certain number of epochs work no better than a random choice.

One of the pathways in order to reach a stable and efficient training is to remove the linear layer that follows the attention model as described before. It was empirically discovered that for larger models which are usually applied in practice (i.e. comparable with BERT-base or larger) the original SimpleTRON architecture shows performance lagging behind the vanilla Transformer. Furthermore, removing the linear layer will not allow to set head dimensions other than d_{model}/h, where d_{model} is the model width and h is a head count.



(a) First layer **(b)** Last layer

Figure 4.6 Training evolution of standard deviation of attention outputs. We compare the three models — vanilla Transformer, Simple and Simple-L.

Another option to stabilize the training is to apply an additional skip connection through the whole Transformer block. This allows to achieve a convergence at any model depth, but does not allow to benefit from the bigger number of stacked blocks. The reason for the deeper models to fail on training is that the weights of the SimpleTRON tend to be symmetrical in the deeper

blocks (see Fig. 4.6). Additional skip connections restrain the weight variance from degeneration and therefore induce better inference ability of the model.

Variational kernel failure We suppose that the variational kernel failed due to the normal distribution being a too string prior for a latent distribution; removing it and leaving the block as an ordinary AE allowed the model to begin training. More complex and adaptable priors such as Gaussian mixtures or Vamp[56] may achieve better results.

Chapter 5

Summary

In this chapter we summarize the provided work. We make conclusions about individual efficient Transformer approaches, observe model design trends and discuss their applicability. We also discuss a broader impact of a research on efficient architectures, and a possibility to integrate them into already existing systems. Finally, we assess the proposed architectures, compare them to baseline models and suggest further possible improvements.

Despite the Transformer success in a widest amount of domains, the architecture itself started a new research field on its own—a family of both faster and even more powerful architectures. As we have already seen in the Chapter 3, recent models are a vivid example of a phenomenon,that a reduction of computational costs should not result in a trade-off with performance; on the contrary, a truly efficient approach can benefit in generalization by getting rid of an unrelated signal. We can draw a parallel with gated recurrent networks there, which also increase their performance through gating and "forgetting" an irrelevant information.

In this work, we have tried to review as much of linear attention mechanisms as possible, as well as to propose our own. In the following sections, we will firstly summarize our findings during an analysis of other approaches, and then will compare and discuss the proposed architectures.

5.1 A summary on modern efficient Transformers

Local attention is not weak One of the important observations is that restricting an attention to some local region introduces a bias beneficial for some tasks. This results in some loss in generality and disables a model from being applied on specific tasks, which require a global attention; however, the intuition standing behind is that a lower amount of relations to consider results in lesser amounts of data needed to learn them. This allows these models to perform better in settings with lower amounts of data provided, such as IMDB or Hyperpartisan classification tasks.

What is even more important, these methods can be easily combined with other approaches, especially with long-term oriented attention alternatives. The most powerful reviewed models always include local attention patterns in some form, and this allows them to:

- Back a global approximation mechanism with a precise mechanism for local relationships
- Enable the model for autoregression, since a local attention allows to generate attention scores for an immediate past (like in Transformer-LS; Nyströmformer can also be adapted to causal prediction via A_S, B_S and F_S triangular masks + local attention).

- Refine a global attention mechanism with a dilated sparse attention, which can improve performance if properly tuned for a task.

Good theoretical bounds do not guarantee a good performance Another interesting observation is that positive results given by a theoretical analysis do not guarantee anything in practice. The simplest example is the whole class of kernelized models, which is, in theory, very optimized for an autoregression (constant memory, linear time complexities) but is extremely slow in practice due to the poor parallelization. Another example is sparse models, which can be significantly slowed down by an absence of a low-level optimization for sparse operations.

A proper theoretical justification also tells very little about eventual test scoring. There is an impressive theoretical background standing behind a Performer, but the architecture still underperforms in some settings compared to the vanilla Transformer or concurrent models, sometimes more primitive and based on intuitions.

Efficient models outperform Transformer We have already seen several architectures, which demonstrate significantly better results on a wide variety of tasks compared to a vanilla Transformer. We can interpret it as introducing a certain bias to the model, and since the original model does not make any strong assumptions about input data, the performance gain may be significant—compared with the models with built-in priors, namely recurrent or convolutional architectures.

5.2 A summary on the proposed models

In the current work, we have introduced the two families of kernelized architectures — the Feedforward kernel and the SimpleTRON. Both models showed good results on the LRA benchmark, and both were not that impressive on the AG News dataset. To explain why, we have conducted additional research to support or refute our hypotheses about the performance.

A weight incompatibility may be the first issue — since we transferred initial weights from the vanilla BERT, we were at a disadvantage when comparing our models with competing architectures, which typically employ pretraining from-scratch on huge datasets such as WikiText-103.

Another issue is related to the Feedforward kernel. The hyperbolic approximation derived in the Performer work illustrates, that the proper function to learn probably belongs to the family of exponentials. In our experiments we employ a softplus output function, which we can assume as linear for larger magnitudes, and significantly differing from the functions employed by other kernelized models. By choosing a model architecture and its output according the appropriate prior we can achieve better performance and convergence.

The SimpleTRON model is harder to analyse, since it does not resemble any architecture reviewed in this work. The effect of the weight incompatibility may stronger there, as well as a larger possible output instability. Since we perform a reduction of unnormalized inputs along the sequence dimension, the assumption of a constant input variance and a sufficiency of the $1/\sqrt{L}$ term for normalization may be naive. For example, to enable the model for training, we have rearranged the LayerNorm placement and put it after summation with a residual connection. Both these hypotheses are being supported by the training-from-scratch experiment, where the model outperformed the vanilla Transformer at the shallower configurations, while getting diminishing returns and eventually degradating with the growing depth.

So, based on the obtained results, we can make the following conclusions:

- It is crucial to perform preliminary test on deeper models; The Long Range Arena[51] benchmark itself is not sufficient, since it offers to measure a performance of a shallow model.
- A good theoretical prior, while not being comprehensive about the final model performance on its own, is a good start and an appropriate intuition for a new architecture.

5.3 On research trends and perspectives

A one can find an evolution pattern between the attention factorization paradigms we reviewed in this work. The sparse attention methods were proposed yet in the "Attention is All You Need[7]" and were a primary direction of research for some time, spanning from locally-concentrated mechanisms such as Blockwise[32] or Image[57] Transformers to Sparse Transformers[58] with different patterns dedicated to capture short- and long-spanning dependencies.

The next logical step would be to move away from fixed patterns and try to learn them instead. This can be traced in the token memory approaches such as Set[36], Star[36] and ETC[35] models. Later methods improve the idea by grouping input tokens, namely by k-means[59] or locality-sensitive hashing[60], and performing an attention inside an assigned bin. However, while these models improved results and remained sub-quadratic, they were still local approaches in a sense that they applied an attention only on subsets of input data.

As an orthogonal alternative, low-rank and kernelized methods emerged to project or factorize an attention rather than to sparsify it. These models differ from the previous approaches in a way they view on data—instead of token sequences with relative positional or vector distances, they operate with inputs as matrices. This allowed to employ a rich mathematical basis and give some guarantees about an approximation precision; however, comparing them[40][48][42] with the most recent sparse attention models such as Longformer[33] or BigBid[34], no specific answer can be given on the question whether this approach is practically better.

The most recent and the most performant efficient attention models belong to hybrid approaches, which try to get best from the different methods. For example, we have discussed the Luna[39] model, which combines a Set Transformer-like factorized attention with a "memory tape", or the Long-Short Transformer[43] which backs the global dynamic low-rank projection with a fixed pattern. Along with the Hierarchical Attention[44], all these models share the N-step paradigm of attention approximation, and we suppose that it will remain the dominant approach for a while.

5.4 An impact of efficient architectures

We have already mentioned several reasons to conduct a research on efficient Transformers. However, enabling the model to be applied within other domains is not the only hypothetical contribution.

Accessibility Transformers are extremely cumbersome to train, and their extensive transfer capabilities do not solve the problem completely. The issue is especially thorny once we arrive to the fact, that an extensive development is still promising, what the new generations of GPT models consistently demonstrate. End users, which apply machine learning models for practical tasks and mostly aim to achieve a satisfying performance, are not affected by the issue as much as smaller research communities are, and the latter cannot rely on extensive amounts of experiments to find a better approach.

Introducing faster attention models may significantly help a research team from several directions:

- An efficient model can be quickly deployed and trained to ensure the correctness of a training pipeline.
- They allow to quickly establish a baseline for comparison on a new task.
- They can achieve better results during a research not directly tied to an architecture.

However, the scientific accessibility is not the only reason. Making the model deployable on end user devices (such as smartphones) may improve their functionality, as well as enable them for federated learning.

Upgrading existing pipelines Most of reviewed architectures more or less demonstrate a backward compatibility with the original Transformer. That enables them to be used as a drop-in replacement for already deployed Transformers, which can speed up an execution time of a system by performing only a slight fine-tuning, while introducing a performance hit varying from a minor negative to positive.

Appendix A

Training Hyperparameters

Parameter	Classif.	Matching	ListOps
Seq. Length	4000	4000	2000
Batch Size	32	32	32
Training Steps	20 000	15 000	15 000
Optimizer	AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.999$)		
Base LR	0.05	0.05	0.005
Weight Decay	0.1	0.1	0.1
Warmup Steps	8000	8000	1000
Schedule	Base LR * Warmup * Sqrt Decay		
Warmup Mul.	$min(1, \text{Current Step}/\text{Warmup Steps})$		
Sqrt Decay Mul.	$1/\sqrt{max(\text{Current Step}, \text{Warmup Steps})}$		
Loss	CCE		
Blocks	4	4	6
Heads	4	4	8
Hidden dim.	256	128	512
QKV dim.	256	128	512
MLP dim.	1024	512	2048
Dropout	0.1	0.1	0.1
Activation	GELU	GELU (ReLU in output)	GELU
Pooling	CLS	CLS	CLS
Pos. encoding	Learnable	Learnable	Learnable

Table A.1 Hyperparameters used for the LRA experiments.

Parameter	**Coarse**	**Fine**
Seq. Length	512	512
Batch Size	32	32
Training Steps	10 500	10 500
Optimizer	AdamW ($\beta_1 = 0.9$, $\beta_2 = 0.999$)	
Base LR	5e−5	5e−6
Layer-Wise LR Decay	0.9	0.9
Weight Decay	1e−4	1e−4
Warmup Steps	-	525
Schedule	Base LR	Base*Warmup*Sqrt
Warmup Mul.	$min(1, \text{Current Step}/\text{Warmup Steps})$	
Loss	CCE	
Blocks	12	
Heads	12	
Hidden dim.	768	
QKV dim.	768	
MLP dim.	3072	
Dropout	0.2	0.2
Activation	GELU	
Pooling	CLS	
Pos. encoding	Learnable	

Table A.2 Hyperparameters used for the AG experiments. "Coarse" stage is tuning only attention weights, output layers and embedddings. "Fine" is a subsequent tuning of the whole model.

Bibliography

1. HOCHREITER, Sepp; SCHMIDHUBER, Jürgen. Long Short-term Memory. *Neural computation*. 1997, vol. 9, pp. 1735–80. Available from DOI: 10.1162/neco.1997.9.8.1735.
2. GREFF, Klaus; SRIVASTAVA, Rupesh Kumar; KOUTNIK, Jan; STEUNEBRINK, Bas R.; SCHMIDHUBER, Jürgen. LSTM: A Search Space Odyssey. *CoRR*. 2015, vol. abs/1503.04069. Available from arXiv: 1503.04069.
3. SUTSKEVER, Ilya; VINYALS, Oriol; LE, Quoc V. Sequence to Sequence Learning with Neural Networks. *CoRR*. 2014, vol. abs/1409.3215. Available from arXiv: 1409.3215.
4. BAHDANAU, Dzmitry; CHO, Kyunghyun; BENGIO, Yoshua. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*. 2014.
5. GRAVES, Alex; WAYNE, Greg; DANIHELKA, Ivo. Neural Turing Machines. *CoRR*. 2014, vol. abs/1410.5401. Available from arXiv: 1410.5401.
6. LUONG, Minh-Thang; PHAM, Hieu; MANNING, Christopher D. Effective Approaches to Attention-based Neural Machine Translation. *CoRR*. 2015, vol. abs/1508.04025. Available from arXiv: 1508.04025.
7. VASWANI, Ashish; SHAZEER, Noam; PARMAR, Niki; USZKOREIT, Jakob; JONES, Llion; GOMEZ, Aidan N.; KAISER, Lukasz; POLOSUKHIN, Illia. Attention Is All You Need. *CoRR*. 2017, vol. abs/1706.03762. Available from arXiv: 1706.03762.
8. DEVLIN, Jacob; CHANG, Ming-Wei; LEE, Kenton; TOUTANOVA, Kristina. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR*. 2018, vol. abs/1810.04805. Available from arXiv: 1810.04805.
9. RADFORD, Alec; NARASIMHAN, Karthik; SALIMANS, Tim; SUTSKEVER, Ilya. Improving language understanding by generative pre-training. 2018.
10. RADFORD, Alec; WU, Jeffrey; CHILD, Rewon; LUAN, David; AMODEI, Dario; SUTSKEVER, Ilya. Language Models are Unsupervised Multitask Learners. 2018. Available also from: https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf.
11. BROWN, Tom B.; MANN, Benjamin; RYDER, Nick; SUBBIAH, Melanie; KAPLAN, Jared; DHARIWAL, Prafulla; NEELAKANTAN, Arvind; SHYAM, Pranav; SASTRY, Girish; ASKELL, Amanda; AGARWAL, Sandhini; HERBERT-VOSS, Ariel; KRUEGER, Gretchen; HENIGHAN, Tom; CHILD, Rewon; RAMESH, Aditya; ZIEGLER, Daniel M.; WU, Jeffrey; WINTER, Clemens; HESSE, Christopher; CHEN, Mark; SIGLER, Eric; LITWIN, Mateusz; GRAY, Scott; CHESS, Benjamin; CLARK, Jack; BERNER, Christopher; MCCANDLISH, Sam; RADFORD, Alec; SUTSKEVER, Ilya; AMODEI, Dario. Language Models are Few-Shot Learners. *CoRR*. 2020, vol. abs/2005.14165. Available from arXiv: 2005.14165.

12. DAI, Zihang; YANG, Zhilin; YANG, Yiming; CARBONELL, Jaime G.; LE, Quoc V.; SALAKHUTDINOV, Ruslan. Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context. *CoRR*. 2019, vol. abs/1901.02860. Available from arXiv: 1901.02860.
13. DOSOVITSKIY, Alexey; BEYER, Lucas; KOLESNIKOV, Alexander; WEISSENBORN, Dirk; ZHAI, Xiaohua; UNTERTHINER, Thomas; DEHGHANI, Mostafa; MINDERER, Matthias; HEIGOLD, Georg; GELLY, Sylvain; USZKOREIT, Jakob; HOULSBY, Neil. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *CoRR*. 2020, vol. abs/2010.11929. Available from arXiv: 2010.11929.
14. CARION, Nicolas; MASSA, Francisco; SYNNAEVE, Gabriel; USUNIER, Nicolas; KIRILLOV, Alexander; ZAGORUYKO, Sergey. End-to-End Object Detection with Transformers. *CoRR*. 2020, vol. abs/2005.12872. Available from arXiv: 2005.12872.
15. SHI, Yangyang; WANG, Yongqiang; WU, Chunyang; YEH, Ching-Feng; CHAN, Julian; ZHANG, Frank; LE, Duc; SELTZER, Michael L. Emformer: Efficient Memory Transformer Based Acoustic Model For Low Latency Streaming Speech Recognition. *CoRR*. 2020, vol. abs/2010.10759. Available from arXiv: 2010.10759.
16. CHEN, Lili; LU, Kevin; RAJESWARAN, Aravind; LEE, Kimin; GROVER, Aditya; LASKIN, Michael; ABBEEL, Pieter; SRINIVAS, Aravind; MORDATCH, Igor. Decision Transformer: Reinforcement Learning via Sequence Modeling. *CoRR*. 2021, vol. abs/2106.01345. Available from arXiv: 2106.01345.
17. TROCKMAN, Asher; KOLTER, J. Zico. Patches Are All You Need? *CoRR*. 2022, vol. abs/2201.09792. Available from arXiv: 2201.09792.
18. *Transformer model for language understanding* [https://www.tensorflow.org/text/tutorials/transformer]. [N.d.]. Accessed: 23-04-2022.
19. VIG, Jesse. A Multiscale Visualization of Attention in the Transformer Model. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. Florence, Italy: Association for Computational Linguistics, 2019, pp. 37–42. Available from DOI: 10.18653/v1/P19-3007.
20. BOJANOWSKI, Piotr; GRAVE, Edouard; JOULIN, Armand; MIKOLOV, Tomas. Enriching Word Vectors with Subword Information. *arXiv preprint arXiv:1607.04606*. 2016.
21. LIU, Yinhan; OTT, Myle; GOYAL, Naman; DU, Jingfei; JOSHI, Mandar; CHEN, Danqi; LEVY, Omer; LEWIS, Mike; ZETTLEMOYER, Luke; STOYANOV, Veselin. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *CoRR*. 2019, vol. abs/1907.11692. Available from arXiv: 1907.11692.
22. JOSHI, Mandar; CHEN, Danqi; LIU, Yinhan; WELD, Daniel S.; ZETTLEMOYER, Luke; LEVY, Omer. SpanBERT: Improving Pre-training by Representing and Predicting Spans. *CoRR*. 2019, vol. abs/1907.10529. Available from arXiv: 1907.10529.
23. CLARK, Kevin; LUONG, Minh-Thang; LE, Quoc V.; MANNING, Christopher D. ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators. *CoRR*. 2020, vol. abs/2003.10555. Available from arXiv: 2003.10555.
24. SANH, Victor; DEBUT, Lysandre; CHAUMOND, Julien; WOLF, Thomas. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*. 2019, vol. abs/1910.01108. Available from arXiv: 1910.01108.
25. LAN, Zhenzhong; CHEN, Mingda; GOODMAN, Sebastian; GIMPEL, Kevin; SHARMA, Piyush; SORICUT, Radu. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. *CoRR*. 2019, vol. abs/1909.11942. Available from arXiv: 1909.11942.
26. RAFFEL, Colin; SHAZEER, Noam; ROBERTS, Adam; LEE, Katherine; NARANG, Sharan; MATENA, Michael; ZHOU, Yanqi; LI, Wei; LIU, Peter J. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *CoRR*. 2019, vol. abs/1910.10683. Available from arXiv: 1910.10683.

27. HE, Kaiming; CHEN, Xinlei; XIE, Saining; LI, Yanghao; DOLLÁR, Piotr; GIRSHICK, Ross B. Masked Autoencoders Are Scalable Vision Learners. *CoRR*. 2021, vol. abs/2111.06377. Available from arXiv: 2111.06377.

28. LU, Kevin; GROVER, Aditya; ABBEEL, Pieter; MORDATCH, Igor. Pretrained Transformers as Universal Computation Engines. *CoRR*. 2021, vol. abs/2103.05247. Available from arXiv: 2103.05247.

29. YUN, Chulhee; BHOJANAPALLI, Srinadh; RAWAT, Ankit Singh; REDDI, Sashank J.; KUMAR, Sanjiv. Are Transformers universal approximators of sequence-to-sequence functions? *CoRR*. 2019, vol. abs/1912.10077. Available from arXiv: 1912.10077.

30. PÉREZ, Jorge; MARINKOVIC, Javier; BARCELÓ, Pablo. On the Turing Completeness of Modern Neural Network Architectures. *CoRR*. 2019, vol. abs/1901.03429. Available from arXiv: 1901.03429.

31. TAY, Yi; DEHGHANI, Mostafa; BAHRI, Dara; METZLER, Donald. Efficient Transformers: A Survey. *CoRR*. 2020, vol. abs/2009.06732. Available from arXiv: 2009.06732.

32. QIU, Jiezhong; MA, Hao; LEVY, Omer; YIH, Scott Wen-tau; WANG, Sinong; TANG, Jie. *Blockwise Self-Attention for Long Document Understanding*. 2020. Available also from: https://openreview.net/forum?id=H1gpET4YDB.

33. BELTAGY, Iz; PETERS, Matthew E.; COHAN, Arman. Longformer: The Long-Document Transformer. *CoRR*. 2020, vol. abs/2004.05150. Available from arXiv: 2004.05150.

34. ZAHEER, Manzil; GURUGANESH, Guru; DUBEY, Avinava; AINSLIE, Joshua; ALBERTI, Chris; ONTAÑÓN, Santiago; PHAM, Philip; RAVULA, Anirudh; WANG, Qifan; YANG, Li; AHMED, Amr. Big Bird: Transformers for Longer Sequences. *CoRR*. 2020, vol. abs/2007.14062. Available from arXiv: 2007.14062.

35. AINSLIE, Joshua; ONTAÑÓN, Santiago; ALBERTI, Chris; PHAM, Philip; RAVULA, Anirudh; SANGHAI, Sumit. ETC: Encoding Long and Structured Data in Transformers. *CoRR*. 2020, vol. abs/2004.08483. Available from arXiv: 2004.08483.

36. GUO, Qipeng; QIU, Xipeng; LIU, Pengfei; SHAO, Yunfan; XUE, Xiangyang; ZHANG, Zheng. Star-Transformer. *CoRR*. 2019, vol. abs/1902.09113. Available from arXiv: 1902.09113.

37. LEE, Juho; LEE, Yoonho; KIM, Jungtaek; KOSIOREK, Adam R.; CHOI, Seungjin; TEH, Yee Whye. Set Transformer. *CoRR*. 2018, vol. abs/1810.00825. Available from arXiv: 1810.00825.

38. ZHANG, Hang; GONG, Yeyun; SHEN, Yelong; LI, Weisheng; LV, Jiancheng; DUAN, Nan; CHEN, Weizhu. Poolingformer: Long Document Modeling with Pooling Attention. *CoRR*. 2021, vol. abs/2105.04371. Available from arXiv: 2105.04371.

39. MA, Xuezhe; KONG, Xiang; WANG, Sinong; ZHOU, Chunting; MAY, Jonathan; MA, Hao; ZETTLEMOYER, Luke. Luna: Linear Unified Nested Attention. *CoRR*. 2021, vol. abs/2106.01540. Available from arXiv: 2106.01540.

40. WANG, Sinong; LI, Belinda Z.; KHABSA, Madian; FANG, Han; MA, Hao. Linformer: Self-Attention with Linear Complexity. *CoRR*. 2020, vol. abs/2006.04768. Available from arXiv: 2006.04768.

41. LEE-THORP, James; AINSLIE, Joshua; ECKSTEIN, Ilya; ONTAÑÓN, Santiago. FNet: Mixing Tokens with Fourier Transforms. *CoRR*. 2021, vol. abs/2105.03824. Available from arXiv: 2105.03824.

42. XIONG, Yunyang; ZENG, Zhanpeng; CHAKRABORTY, Rudrasis; TAN, Mingxing; FUNG, Glenn; LI, Yin; SINGH, Vikas. Nyströmformer: A Nyström-Based Algorithm for Approximating Self-Attention. *CoRR*. 2021, vol. abs/2102.03902. Available from arXiv: 2102.03902.

43. ZHU, Chen; PING, Wei; XIAO, Chaowei; SHOEYBI, Mohammad; GOLDSTEIN, Tom; ANANDKUMAR, Anima; CATANZARO, Bryan. Long-Short Transformer: Efficient Transformers for Language and Vision. *CoRR*. 2021, vol. abs/2107.02192. Available from arXiv: 2107.02192.

44. ZHU, Zhenhai; SORICUT, Radu. H-Transformer-1D: Fast One-Dimensional Hierarchical Attention for Sequences. *CoRR*. 2021, vol. abs/2107.11906. Available from arXiv: 2107.11906.

45. HUTCHINS, DeLesley; SCHLAG, Imanol; WU, Yuhuai; DYER, Ethan; NEYSHABUR, Behnam. *Block-Recurrent Transformers*. arXiv, 2022. Available from DOI: 10.48550/ARXIV.2203.07852.

46. TSAI, Yao-Hung Hubert; BAI, Shaojie; YAMADA, Makoto; MORENCY, Louis-Philippe; SALAKHUTDINOV, Ruslan. Transformer Dissection: An Unified Understanding for Transformer's Attention via the Lens of Kernel. *CoRR*. 2019, vol. abs/1908.11775. Available from arXiv: 1908.11775.

47. KATHAROPOULOS, Angelos; VYAS, Apoorv; PAPPAS, Nikolaos; FLEURET, François. Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention. *CoRR*. 2020, vol. abs/2006.16236. Available from arXiv: 2006.16236.

48. CHOROMANSKI, Krzysztof; LIKHOSHERSTOV, Valerii; DOHAN, David; SONG, Xingyou; GANE, Andreea; SARLÓS, Tamás; HAWKINS, Peter; DAVIS, Jared; MOHIUDDIN, Afroz; KAISER, Lukasz; BELANGER, David; COLWELL, Lucy J.; WELLER, Adrian. Rethinking Attention with Performers. *CoRR*. 2020, vol. abs/2009.14794. Available from arXiv: 2009.14794.

49. CHOWDHURY, Sankalan Pal; SOLOMOU, Adamos; DUBEY, Avinava; SACHAN, Mrinmaya. On Learning the Transformer Kernel. *CoRR*. 2021, vol. abs/2110.08323. Available from arXiv: 2110.08323.

50. LE, Quoc V.; SARLOS, Tamas; SMOLA, Alex. Fastfood-computing hilbert space expansions in loglinear time. In: *International Conference on Machine Learning*. 2013. Available also from: http://www-cs.stanford.edu/~quocle/LeSarlosSmola_ICML13.pdf.

51. TAY, Yi; DEHGHANI, Mostafa; ABNAR, Samira; SHEN, Yikang; BAHRI, Dara; PHAM, Philip; RAO, Jinfeng; YANG, Liu; RUDER, Sebastian; METZLER, Donald. Long Range Arena : A Benchmark for Efficient Transformers. In: *International Conference on Learning Representations*. 2021. Available also from: https://openreview.net/forum?id=qVyeW-grC2k.

52. YUN, Chulhee; CHANG, Yin-Wen; BHOJANAPALLI, Srinadh; RAWAT, Ankit Singh; REDDI, Sashank J.; KUMAR, Sanjiv. O(n) Connections are Expressive Enough: Universal Approximability of Sparse Transformers. *CoRR*. 2020, vol. abs/2006.04862. Available from arXiv: 2006.04862.

53. JIA, Kui; LI, Shuai; WEN, Yuxin; LIU, Tongliang; TAO, Dacheng. Orthogonal Deep Neural Networks. *CoRR*. 2019, vol. abs/1905.05929. Available from arXiv: 1905.05929.

54. SUN, Chi; QIU, Xipeng; XU, Yige; HUANG, Xuanjing. How to Fine-Tune BERT for Text Classification? *CoRR*. 2019, vol. abs/1905.05583. Available from arXiv: 1905.05583.

55. *Huggingface BERT Description* [https://huggingface.co/docs/transformers/model_doc/bert]. [N.d.]. Accessed: 28-04-2022.

56. TOMCZAK, Jakub M.; WELLING, Max. VAE with a VampPrior. *CoRR*. 2017, vol. abs/1705.07120. Available from arXiv: 1705.07120.

57. PARMAR, Niki; VASWANI, Ashish; USZKOREIT, Jakob; KAISER, Lukasz; SHAZEER, Noam; KU, Alexander. Image Transformer. *CoRR*. 2018, vol. abs/1802.05751. Available from arXiv: 1802.05751.

58. CHILD, Rewon; GRAY, Scott; RADFORD, Alec; SUTSKEVER, Ilya. Generating Long Sequences with Sparse Transformers. *CoRR*. 2019, vol. abs/1904.10509. Available from arXiv: 1904.10509.

59. ROY, Aurko; SAFFAR, Mohammad; VASWANI, Ashish; GRANGIER, David. Efficient Content-Based Sparse Attention with Routing Transformers. *CoRR*. 2020, vol. abs/2003.05997. Available from arXiv: 2003.05997.

60. KITAEV, Nikita; KAISER, Lukasz; LEVSKAYA, Anselm. Reformer: The Efficient Transformer. *CoRR*. 2020, vol. abs/2001.04451. Available from arXiv: 2001.04451.

Attached Medium

- `readme.txt` — medium contents description
- `tex` — LaTeX sources directory
- `src`
  - `download.sh` — download script for data
  - `example_ag.ipynb` — Example notebook with an AG experiment
  - `example_cls.ipynb` — Example notebook with a classification experiment on LRA
  - `example_lops.ipynb` — Example notebook with a ListOps experiment on LRA
  - `example_ret.ipynb` — Example notebook with a retrieval experiment on LRA
  - `ag` — AG setup sources
    - `notebooks` — Colab notebooks with AG experiments drafts
  - `lra` — LRA setup sources
    - `notebooks` — Colab notebooks with LRA experiments drafts
- `yorshula-master.pdf` — the work in the .pdf format