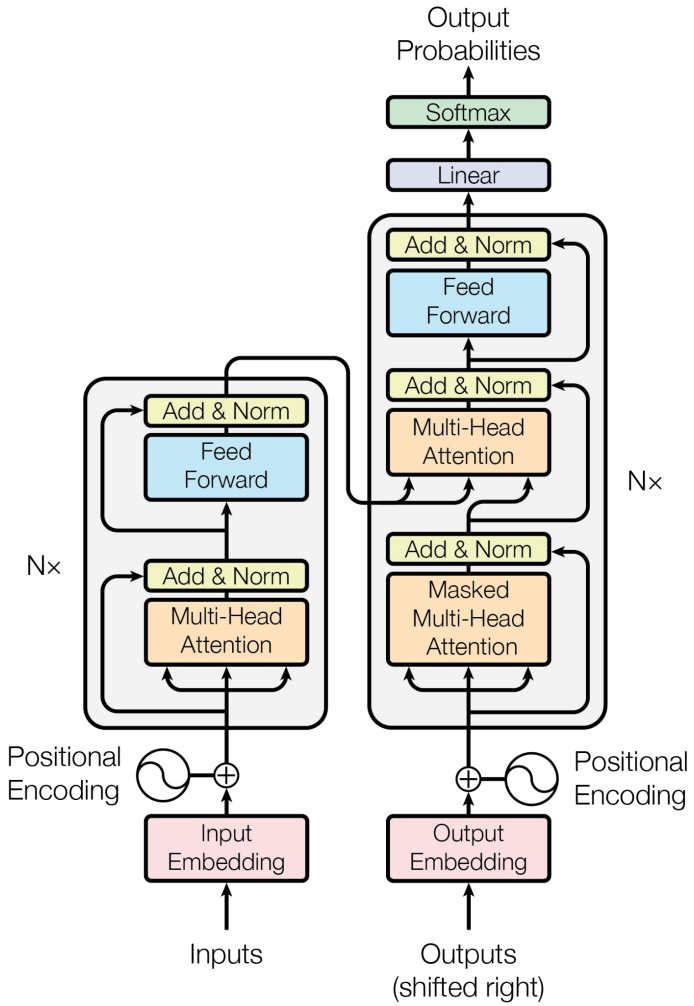
Attention Mechanisms

Wangshu Zhu (wz2708)
Berk Yilmaz (by2385)
Dan Harvey (dyh2111)

EECS E6691 Advanced Deep Learning, 2025 Spring

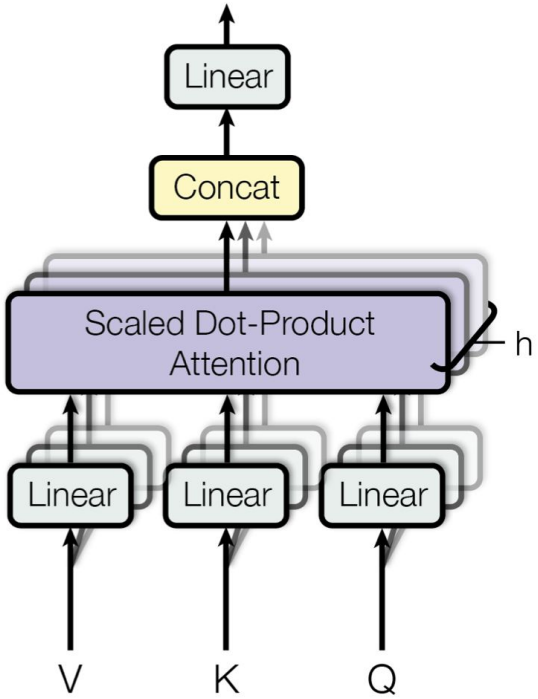


Outline of the Presentation

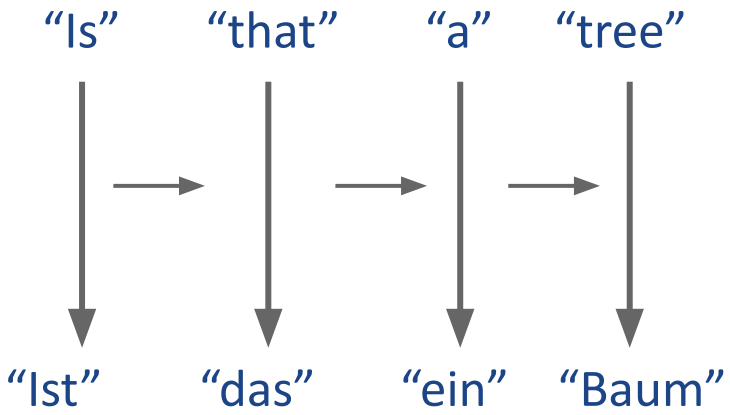
History
"Why do we need attention?"
Challenges in Seq2Se2 Models
Basic Mechanism of Attention
Additive Attention

Self Attention
Multi-Headed Attention
Masked Multi-Headed Attention

SoTA Methods



Brief History



Brief History

In 2014,

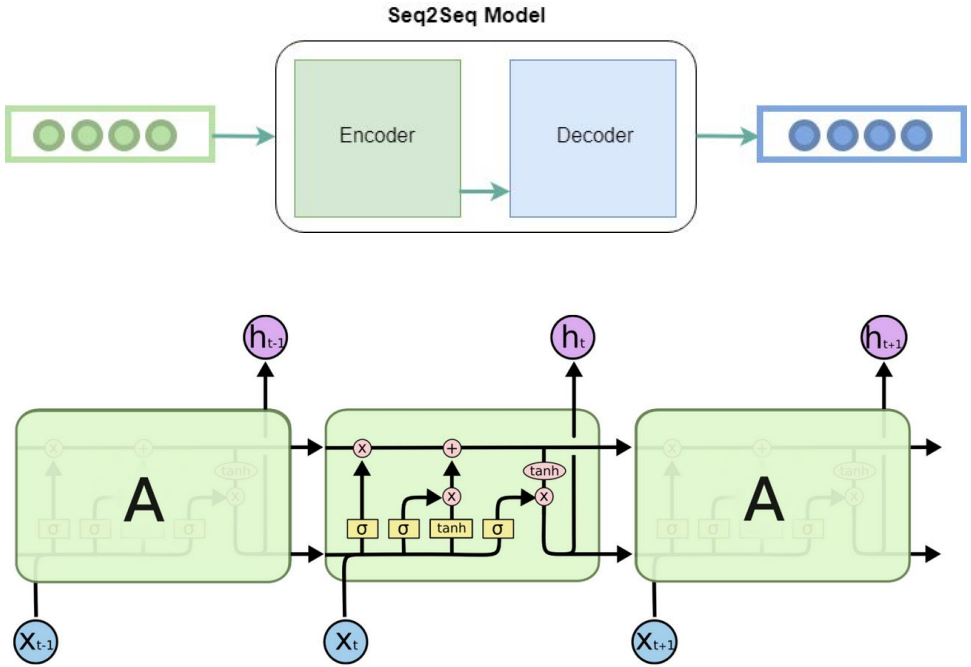


Figure 1. LSTM Block

source : https://sh-tsang.medium.com/review-seq2seq-sequence-to-sequence-learning-with-neural-networks-bcb84071a670

Sequence to Sequence Learning with Neural Networks

Ilya Sutskever
Google
ilyasu@google.com

Oriol Vinyals
Google
vinyals@google.com

Quoc V. Le
Google
qvl@google.com

Abstract

Deep Neural Networks (DNNs) are powerful models that have achieved excellent performance on difficult learning tasks. Although DNNs work well whenever large labeled training sets are available, they cannot be used to map sequences to sequences. In this paper, we present a general end-to-end approach to sequence learning that makes minimal assumptions on the sequence structure. Our method uses a multilayered Long Short-Term Memory (LSTM) to map the input sequence to a vector of a fixed dimensionality, and then another deep LSTM to decode the target sequence from the vector. Our main result is that on an English to French translation task from the WMT'14 dataset, the translations produced by the LSTM achieve a BLEU score of 34.8 on the entire test set, where the LSTM's BLEU score was penalized on out-of-vocabulary words. Additionally, the LSTM did not have difficulty on long sentences. For comparison, a phrase-based SMT system achieves a BLEU score of 33.3 on the same dataset. When we used the LSTM to rerank the 1000 hypotheses produced by the aforementioned SMT system, its BLEU score increases to 36.5, which is close to the previous best result on this task. The LSTM also learned sensible phrase and sentence representations that are sensitive to word order and are relatively invariant to the active and the passive voice. Finally, we found that reversing the order of the words in all source sentences (but not target sentences) improved the LSTM's performance markedly, because doing so introduced many short term dependencies between the source and the target sentence which made the optimization problem easier.

1 Introduction

Deep Neural Networks (DNNs) are extremely powerful machine learning models that achieve excellent performance on difficult problems such as speech recognition [13, 7] and visual object recognition [19, 6, 21, 20]. DNNs are powerful because they can perform arbitrary parallel computation for a modest number of steps. A surprising example of the power of DNNs is their ability to sort N N-bit numbers using only 2 hidden layers of quadratic size [27]. So, while neural networks are related to conventional statistical models, they learn an intricate computation. Furthermore, large DNNs can be trained with supervised backpropagation whenever the labeled training set has enough information to specify the network's parameters. Thus, if there exists a parameter setting of a large DNN that achieves good results (for example, because humans can solve the task very rapidly), supervised backpropagation will find these parameters and solve the problem.

Despite their flexibility and power, DNNs can only be applied to problems whose inputs and targets can be sensibly encoded with vectors of fixed dimensionality. It is a significant limitation, since many important problems are best expressed with sequences whose lengths are not known a-priori. For example, speech recognition and machine translation are sequential problems. Likewise, question answering can also be seen as mapping a sequence of words representing the question to a

1

Figure 2. Sequence to Sequence Learning with Neural Networks

source : https://arxiv.org/abs/1409.3215

Challenges in Seq2Seq Models

The Encoder Bottleneck

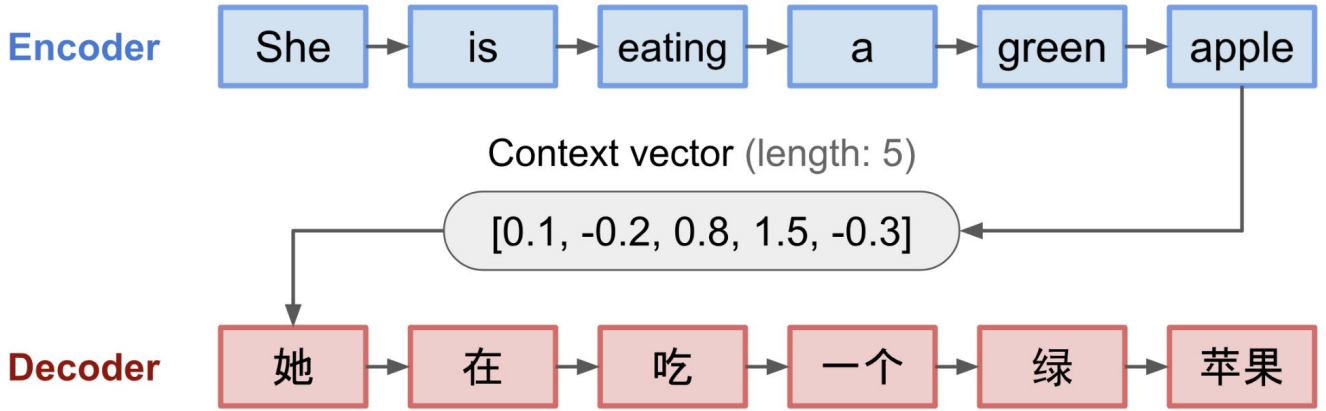


Figure 3. Encoder - Decoder Architecture

source :https://library.fiveable.me/natural-language-processing/unit-8/encoder-decoder-architecture/study-guide/0Ku3zt66xYRvLfvu

Challenges in Seq2Seq Models

Languages are not the same, ambiguous references

English	Turkish
She is gorgeous	**O** çok güzel
He is gorgeous	**O** çok güzel

It requires the words or sentences that come before it to understand its true meaning.

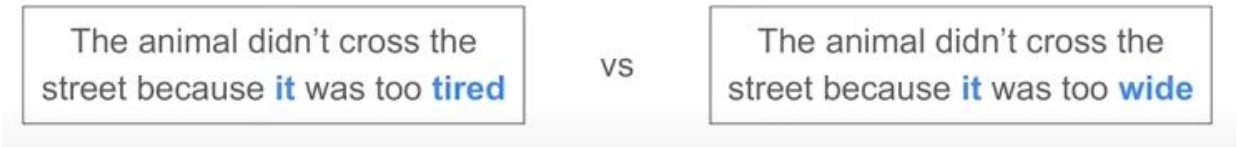
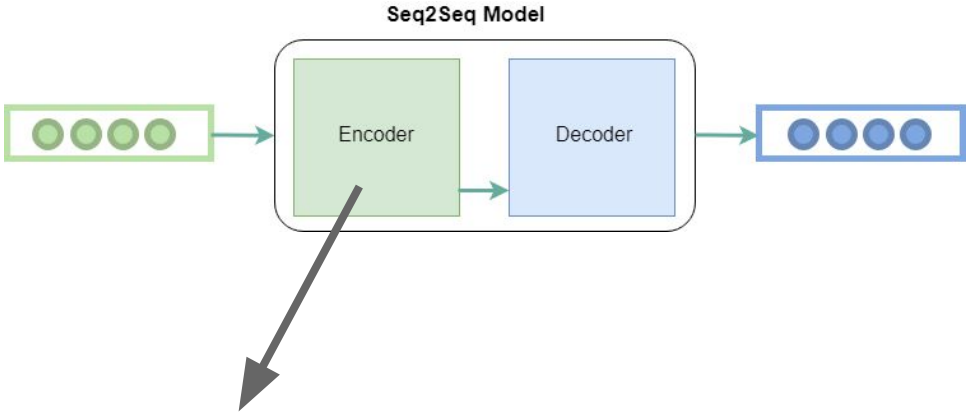


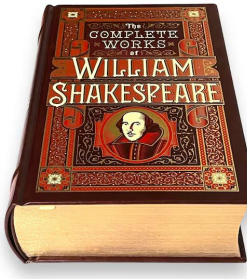
Figure 4. Ambiguous Reference
source :https://www.youtube.com/watch?v=TyU3qyIMWpE&t=72s

Challenges in Seq2Seq Models

The Encoder Bottleneck



Entire input is packed on single vector



Too much information on a single vector!

Soft-Search

Seq2seq models struggle with **long sentences** due to the fixed-length **encoding bottleneck**

The **attention mechanism** solves this by dynamically **focusing on relevant input parts** at each decoding step.

" In this paper, we conjecture that the use of a fixed-length vector is a bottleneck in improving the performance of this basic encoder–decoder architecture, and propose to extend this by allowing a model to automatically **(soft-)search** for parts of a source sentence that are relevant to predicting a target word, without having to form these parts as a hard segment explicitly. "

Published as a conference paper at ICLR 2015

NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

Dzmitry Bahdanau
Jacobs University Bremen, Germany

KyungHyun Cho **Yoshua Bengio***
Université de Montréal

ABSTRACT

Neural machine translation is a recently proposed approach to machine translation. Unlike the traditional statistical machine translation, the neural machine translation aims at building a single neural network that can be jointly tuned to maximize the translation performance. The models proposed recently for neural machine translation often belong to a family of encoder–decoders and encode a source sentence into a fixed-length vector from which a decoder generates a translation. In this paper, we conjecture that the use of a fixed-length vector is a bottleneck in improving the performance of this basic encoder–decoder architecture, and propose to extend this by allowing a model to automatically (soft-)search for parts of a source sentence that are relevant to predicting a target word, without having to form these parts as a hard segment explicitly. With this new approach, we achieve a translation performance comparable to the existing state-of-the-art phrase-based system on the task of English-to-French translation. Furthermore, qualitative analysis reveals that the (soft-)alignments found by the model agree well with our intuition.

1 INTRODUCTION

Neural machine translation is a newly emerging approach to machine translation, recently proposed by Kalchbrenner and Blunsom (2013), Sutskever *et al.* (2014) and Cho *et al.* (2014b). Unlike the traditional phrase-based translation system (see, e.g., Koehn *et al.*, 2003) which consists of many small sub-components that are tuned separately, neural machine translation attempts to build and train a single, large neural network that reads a sentence and outputs a correct translation.

Most of the proposed neural machine translation models belong to a family of *encoder–decoders* (Sutskever *et al.*, 2014; Cho *et al.*, 2014a), with an encoder and a decoder for each language, or involve a language-specific encoder applied to each sentence whose outputs are then compared (Hermann and Blunsom, 2014). An encoder neural network reads and encodes a source sentence into a fixed-length vector. A decoder then outputs a translation from the encoded vector. The whole encoder–decoder system, which consists of the encoder and the decoder for a language pair, is jointly trained to maximize the probability of a correct translation given a source sentence.

A potential issue with this encoder–decoder approach is that a neural network needs to be able to compress all the necessary information of a source sentence into a fixed-length vector. This may make it difficult for the neural network to cope with long sentences, especially those that are longer than the sentences in the training corpus. Cho *et al.* (2014b) showed that indeed the performance of a basic encoder–decoder deteriorates rapidly as the length of an input sentence increases.

In order to address this issue, we introduce an extension to the encoder–decoder model which learns to align and translate jointly. Each time the proposed model generates a word in a translation, it (soft-)searches for a set of positions in a source sentence where the most relevant information is concentrated. The model then predicts a target word based on the context vectors associated with these source positions and all the previous generated target words.

*CIFAR Senior Fellow

Figure 5. Neural Machine Learning Translation by jointly Learning To Align and Translate
https://arxiv.org/abs/1409.0473

Soft-Search

Older models;

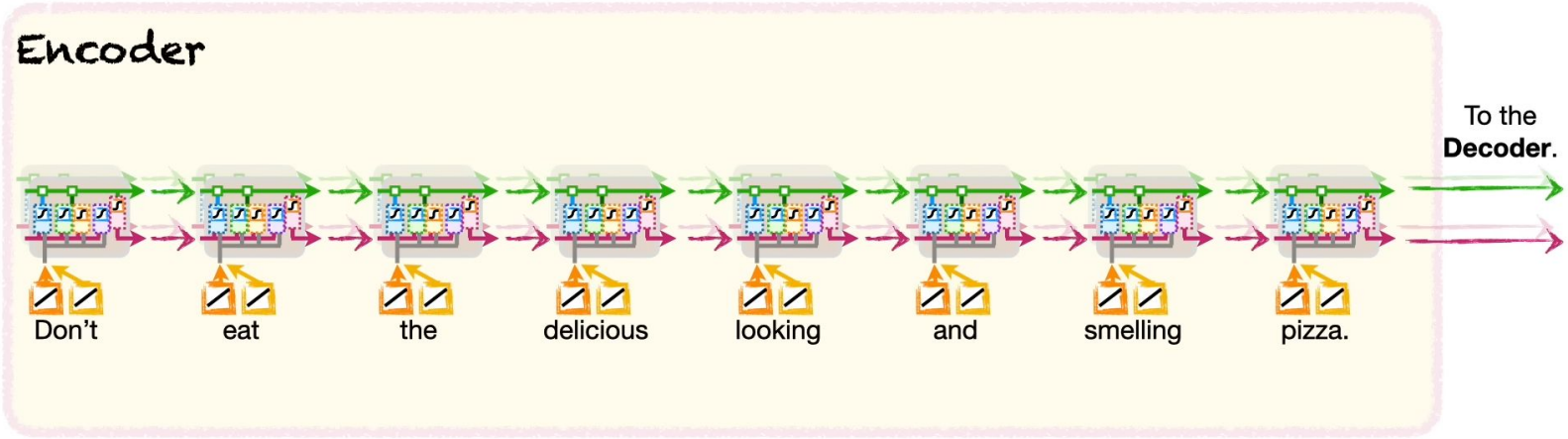


Figure 6. A word sequence
source :https://www.youtube.com/watch?v=PSs6nxngL6k

How does the Attention Mechanism Work?

Attention lets us focus on the most important information while filtering out the rest. This selective focus not only limits what we process but also helps us recognize and concentrate on crucial details in similar future situations

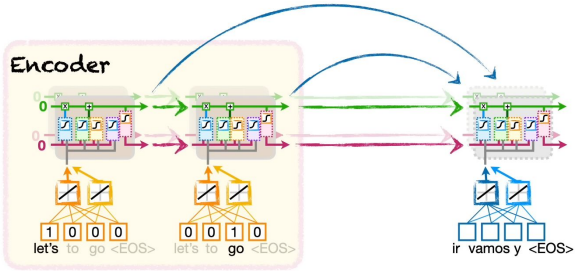


Figure 7. LSTMs compression of input short sentences with Attention
source :https://www.youtube.com/watch?v=PSs6nxngL6k

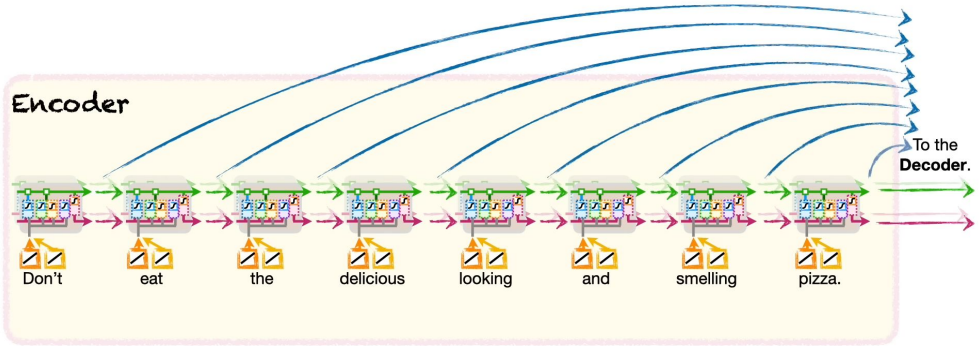


Figure 8.. LSTMs compression of input short sentences with Attention
source :https://www.youtube.com/watch?v=PSs6nxngL6k

Attention Mechanism

Reference Article : https://medium.com/towards-data-science/rethinking-thinking-how-do-attention-mechanisms-actually-work-a6f67d313f99

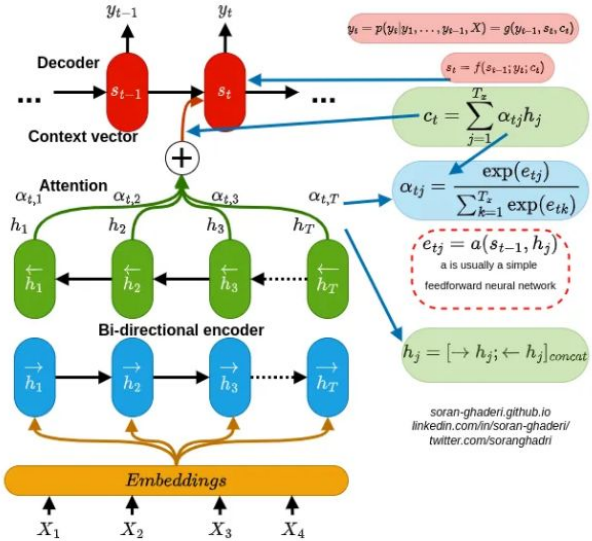


Figure 9. Attention mechanism
source :https://medium.com/towards-data-science/rethinking-thinking-how-do-attention-mechanisms-actually-work-a6f67d313f99

At each step of translating a word, it (soft-)searches for the most relevant information across **various positions in the source sentence**

It then generates **translations for the source word using a context vector** derived jointly from relevant positions and prior words.

Attention Mechanism

RNNsearch is bidirectional

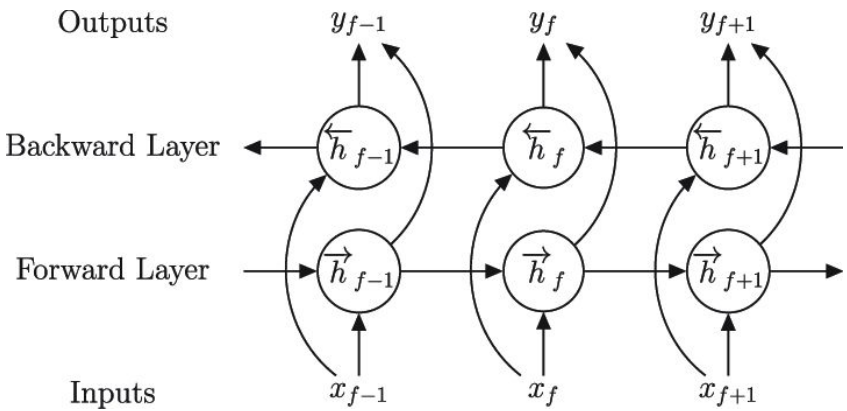
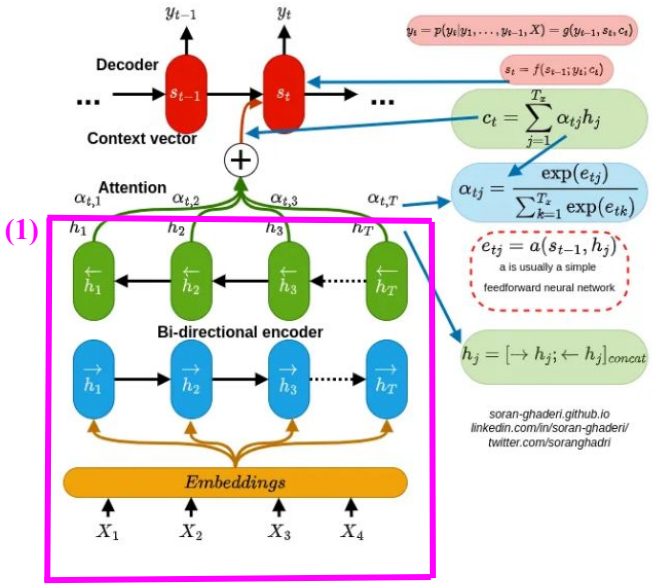


Figure 10. Bidirectional RNN networks
Source : https://www.researchgate.net/figure/Bidirectional-recurrent-neural-network-50_fig4_325519679

Each word or time step in the input sequence, the **encoder generates a hidden state** that incorporates information from both the preceding and following words

General Overlay of Attention Mechanism

Reference Article : *https://medium.com/towards-data-science/rethinking-thinking-how-do-attention-mechanisms-actually-work-a6f67d313f99*



(1)

The inputs **(X1, X2…XT)** are fed into the **forward RNN** to produce the forward hidden states

$$\{\overrightarrow{h_1}, \ldots, \overrightarrow{h_T}\}$$

Forward hidden states

The **backward RNN** processes the input sequence in reverse order, starting from XT and moving to X1 . This generates a sequence of backward hidden states

$$\{\overleftarrow{h_1}, \ldots, \overleftarrow{h_T}\}$$

Backward hidden states

The model produces an annotation for each input Xi by combining the forward hidden state $h1$ and the backward hidden state hT through **concatenation**

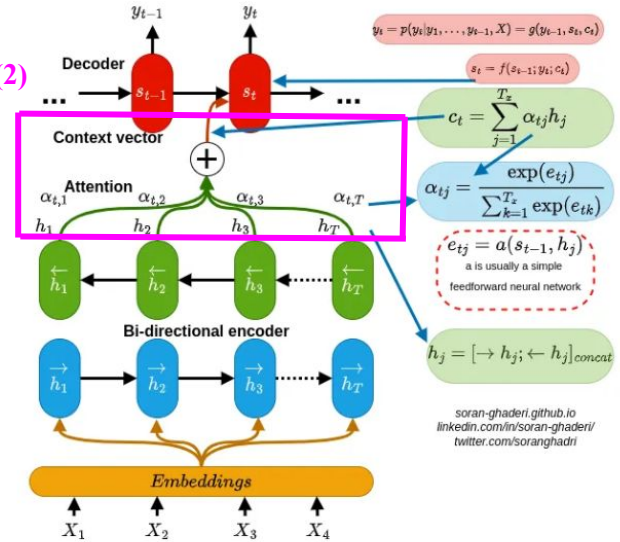
$$h_i = [\overrightarrow{h_i}^\top; \overleftarrow{h_i}]^\top$$

This combined representation is denoted as **h*i***

General Overlay of Attention Mechanism

Reference Article : https://medium.com/towards-data-science/rethinking-thinking-how-do-attention-mechanisms-actually-work-a6f67d313f99

(2)



(2)

The attention block computes the **context vector ct,** which encodes the **relationship between the current output (at time step t) and all the input hidden states (hj)**

$$\boldsymbol{c}_t = \sum_{j=1}^{T} \alpha_{tj} \boldsymbol{h}_j.$$

The context vector c*t* is derived as a weighted sum of the hidden states h*j*

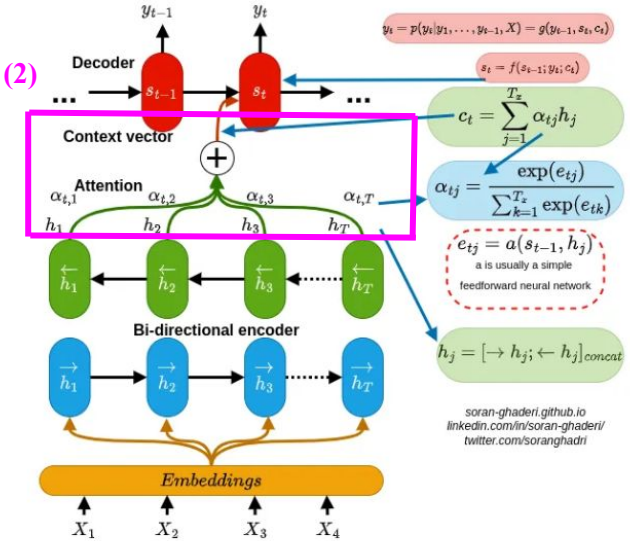
The term α_{tj} represents the attention weight assigned to each annotation **(hidden state)** h_j at time step t

This, indicate the **importance or relevance** of each h_j to the current output being generated at time step t

General Overlay of Attention Mechanism

Reference Article : https://medium.com/towards-data-science/rethinking-thinking-how-do-attention-mechanisms-actually-work-a6f67d313f99

(2)



(2 cont, NOTE)

The attention weights are typically derived using a **scoring function** (e.g., dot product, additive, or multiplicative attention)

However, it can vary

Bahdanau (Additive) Attention
Introduced in 2015 paper

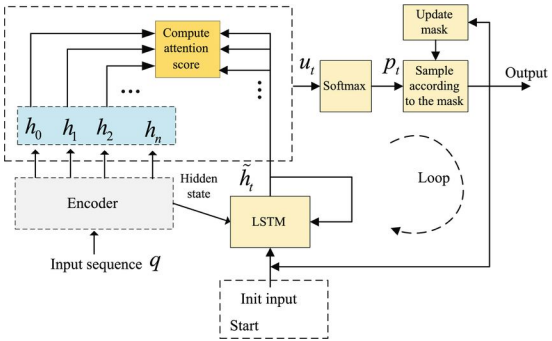
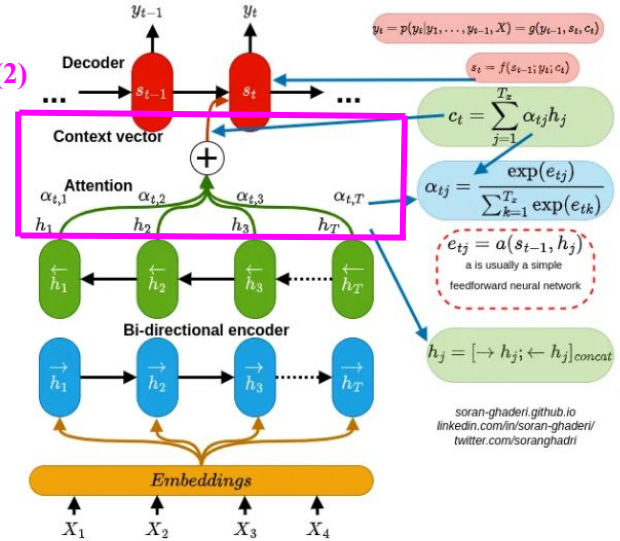


Figure 10. Bahdanau (Additive) Attention
source : https://www.researchgate.net/figure/The-framework-of-additive-attention-mechanism-in-decoder_fig2_362369368

General Overlay of Attention Mechanism

Reference Article : https://medium.com/towards-data-science/rethinking-thinking-how-do-attention-mechanisms-actually-work-a6f67d313f99

(2)



(2 cont, NOTE)

The attention weights are typically derived using a **scoring function** (e.g., dot product, additive, or multiplicative attention)

These will be covered later in the presentation

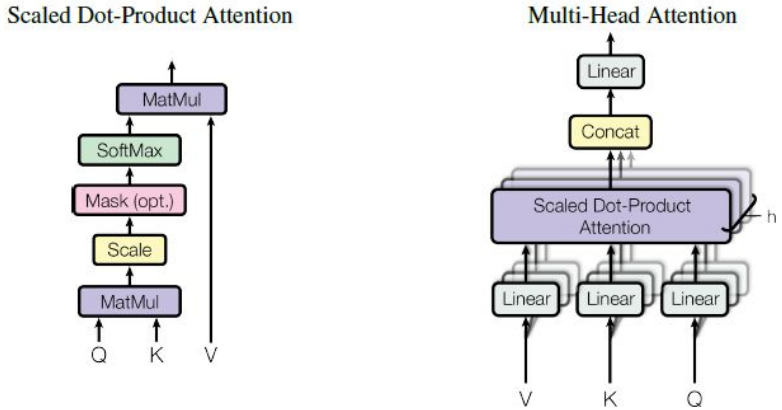
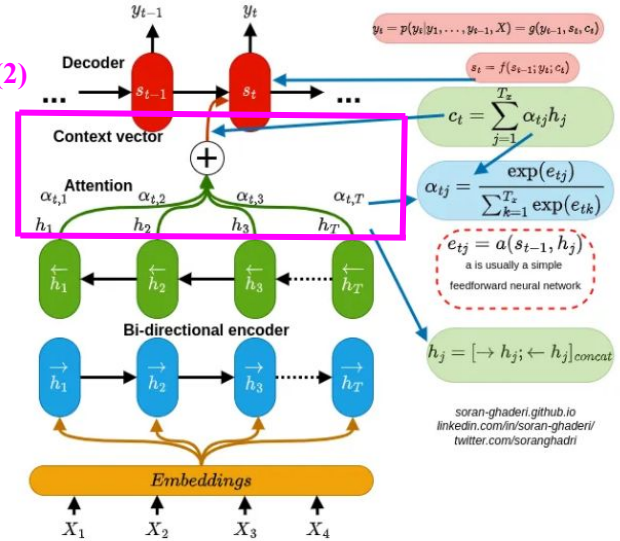


Figure 11. Scaled Dot-Product Attention and Multi-Head Attention
source : https://vitalflux.com/attention-mechanism-in-transformers-examples/

General Overlay of Attention Mechanism Mechanism

Reference Article : https://medium.com/towards-data-science/rethinking-thinking-how-do-attention-mechanisms-actually-work-a6f67d313f99

(2)



(2 cont, NOTE)

Followed by a **softmax normalization** to ensure they sum to 1

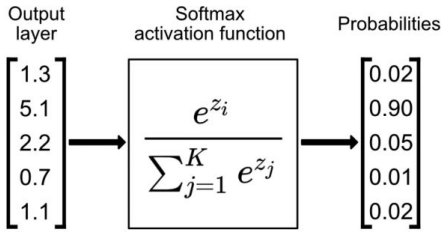


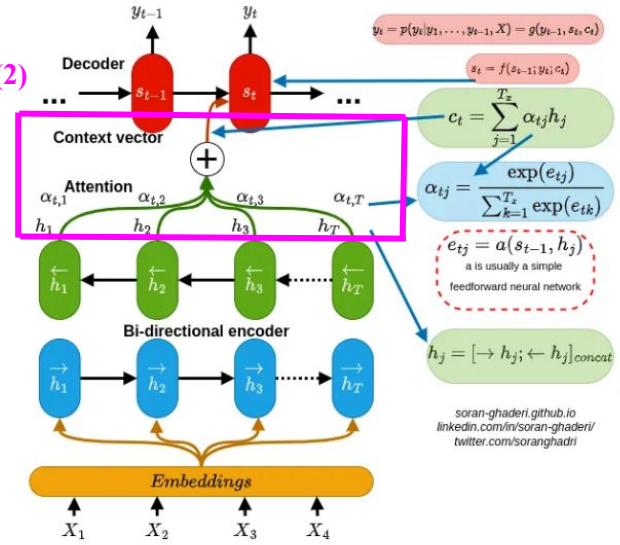
Figure 12. Softmax Function
source : https://www.singlestore.com/blog/a-guide-to-softmax-activation-function/

Softmax Function : https://www.geeksforgeeks.org/the-role-of-softmax-in-neural-networks-detailed-explanation-and-applications/

General Overlay of Attention Mechanism

Reference Article : https://medium.com/towards-data-science/rethinking-thinking-how-do-attention-mechanisms-actually-work-a6f67d313f99

(2)



(2 cont,)

The attention block computes the **context vector c*t*,** which encodes the **relationship between the current output (at time step *t*) and all the input hidden states (h*j*)**

$$\boldsymbol{c}_t = \sum_{j=1}^{T} \alpha_{tj} \boldsymbol{h}_j.$$

The context vector c*t* is derived as a weighted sum of the hidden states h*j*

α*tj* represents the attention weight assigned to each annotation **(hidden state) h*j*** at time step ***t***

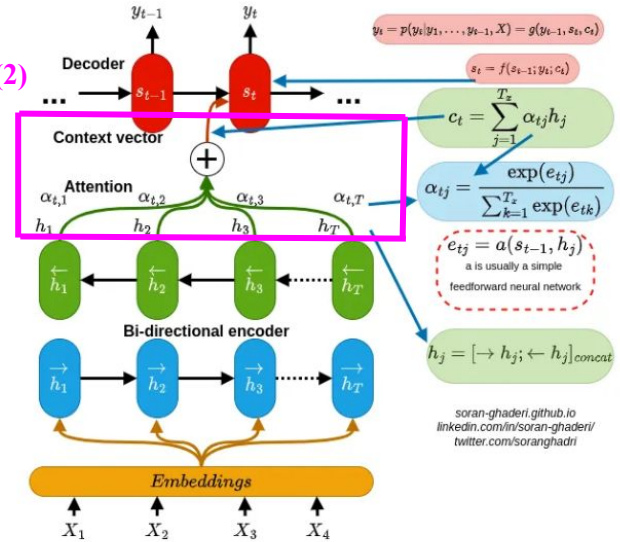
$$\alpha_{tj} = \frac{\exp(e_{tj})}{\sum_{k=1}^{T} \exp(e_{tk})},$$

SOFTMAX

General Overlay of Attention Mechanism

Reference Article : https://medium.com/towards-data-science/rethinking-thinking-how-do-attention-mechanisms-actually-work-a6f67d313f99

(2)



(2 cont,)

$$\alpha_{tj} = \frac{\exp(e_{tj})}{\sum_{k=1}^{T} \exp(e_{tk})},$$

attention weight assigned to each annotation

and e_{tj} can be calculated as

$$e_{tj} = a(\boldsymbol{s}_{t-1}, \boldsymbol{h}_j),$$

s_{t-1} is the previous hidden state of the decoder (or output RNN)

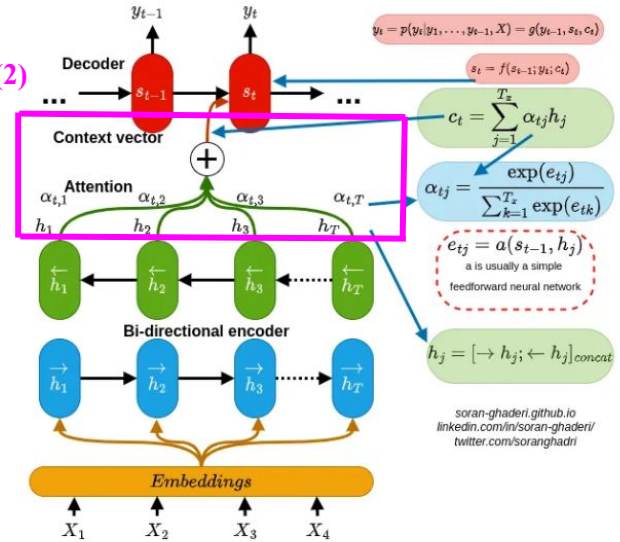
h_j is the annotation (hidden state) from the encoder at time step j

a is the alignment model, which computes the compatibility or relevance between s_{t-1} and h_j

General Overlay of Attention Mechanism

Reference Article : https://medium.com/towards-data-science/rethinking-thinking-how-do-attention-mechanisms-actually-work-a6f67d313f99

(2)



(2 cont,)

$$\alpha_{tj} = \frac{\exp(e_{tj})}{\sum_{k=1}^{T} \exp(e_{tk})},$$

attention weight assigned to each annotation

and e_{tj} can be calculated as

$$e_{tj} = a(\boldsymbol{s}_{t-1}, \boldsymbol{h}_j),$$

s_{t-1} is the previous hidden state of the decoder (or output RNN)

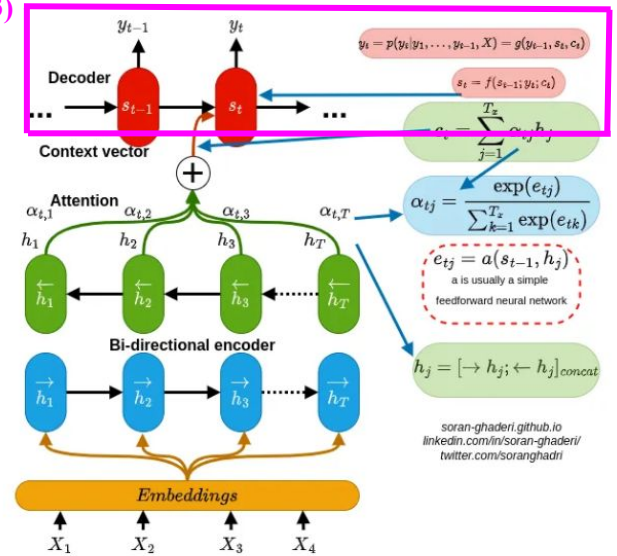
h_j is the annotation (hidden state) from the encoder at time step j

a is the alignment model, which computes the compatibility or relevance between s_{t-1} and h_j

General Overlay of Attention Mechanism

Reference Article : https://medium.com/towards-data-science/rethinking-thinking-how-do-attention-mechanisms-actually-work-a6f67d313f99

(3)



(3)

$$\alpha_{tj} = \frac{\exp(e_{tj})}{\sum_{k=1}^{T} \exp(e_{tk})},$$

attention weight assigned to each annotation

and e_{tj} can be calculated as

$$e_{tj} = a(\boldsymbol{s}_{t-1}, \boldsymbol{h}_j),$$

s_{t-1} is the previous hidden state of the decoder (or output RNN)

h_j is the annotation (hidden state) from the encoder at time step j

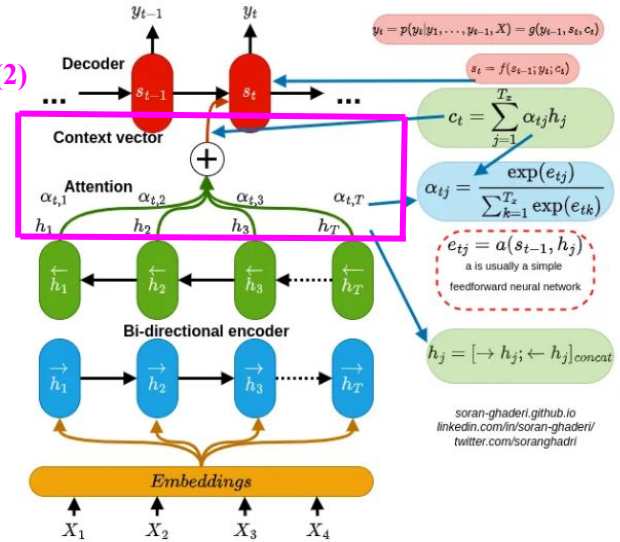
Updating the decoder's hidden state

$$s_i = f(s_{i-1}, y_{i-1}, c_i)$$

General Overlay of Attention Mechanism

Reference Article : https://medium.com/towards-data-science/rethinking-thinking-how-do-attention-mechanisms-actually-work-a6f67d313f99

(2)



(3 cont.)

$$s_i = f(s_{i-1}, y_{i-1}, c_i)$$

Once the hidden state s_i is computed, the model generates the most probable output y_t at the current step

$$p(y_t|y_1, \ldots, y_{t-1}, \boldsymbol{x}) = \text{RNN}(\boldsymbol{c}_t).$$

This is done by applying a ***softmax function*** to the output

Bahdanau (Additive) Model Adoption

Published as a conference paper at ICLR 2015

NEURAL MACHINE TRANSLATION
BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

Dzmitry Bahdanau
Jacobs University Bremen, Germany

KyungHyun Cho **Yoshua Bengio***
Université de Montréal

ABSTRACT

Neural machine translation is a recently proposed approach to machine translation. Unlike the traditional statistical machine translation, the neural machine translation aims at building a single neural network that can be jointly tuned to maximize the translation performance. The models proposed recently for neural machine translation often belong to a family of encoder–decoders and encode a source sentence into a fixed-length vector from which a decoder generates a translation. In this paper, we conjecture that the use of a fixed-length vector is a bottleneck in improving the performance of this basic encoder–decoder architecture, and propose to extend this by allowing a model to automatically (soft-)search for parts of a source sentence that are relevant to predicting a target word, without having to form these parts as a hard segment explicitly. With this new approach, we achieve a translation performance comparable to the existing state-of-the-art phrase-based system on the task of English-to-French translation. Furthermore, qualitative analysis reveals that the (soft-)alignments found by the model agree well with our intuition.

1 INTRODUCTION

Neural machine translation is a newly emerging approach to machine translation, recently proposed by Kalchbrenner and Blunsom (2013), Sutskever *et al.* (2014) and Cho *et al.* (2014b). Unlike the traditional phrase-based translation system (see, e.g., Koehn *et al.*, 2003) which consists of many small sub-components that are tuned separately, neural machine translation attempts to build and train a single, large neural network that reads a sentence and outputs a correct translation.

Most of the proposed neural machine translation models belong to a family of *encoder–decoders* (Sutskever *et al.*, 2014; Cho *et al.*, 2014a), with an encoder and a decoder for each language, or involve a language-specific encoder applied to each sentence whose outputs are then compared (Hermann and Blunsom, 2014). An encoder neural network reads and encodes a source sentence into a fixed-length vector. A decoder then outputs a translation from the encoded vector. The whole encoder–decoder system, which consists of the encoder and the decoder for a language pair, is jointly trained to maximize the probability of a correct translation given a source sentence.

A potential issue with this encoder–decoder approach is that a neural network needs to be able to compress all the necessary information of a source sentence into a fixed-length vector. This may make it difficult for the neural network to cope with long sentences, especially those that are longer than the sentences in the training corpus. Cho *et al.* (2014b) showed that indeed the performance of a basic encoder–decoder deteriorates rapidly as the length of an input sentence increases.

In order to address this issue, we introduce an extension to the encoder–decoder model which learns to align and translate jointly. Each time the proposed model generates a word in a translation, it (soft-)searches for a set of positions in a source sentence where the most relevant information is concentrated. The model then predicts a target word based on the context vectors associated with these source positions and all the previous generated target words.

*CIFAR Senior Fellow

1

Figure 13. Neural Machine Translation by Jointly Learning to Align and Translate
source :https://arxiv.org/abs/1409.0473

After the introduction of the **Bahdanau attention mechanism (also known as additive attention)** in neural machine translation, researchers developed various other attention mechanisms

Despite their differences, most attention mechanisms share two common steps

1- Computation of Attention Distribution on the Inputs

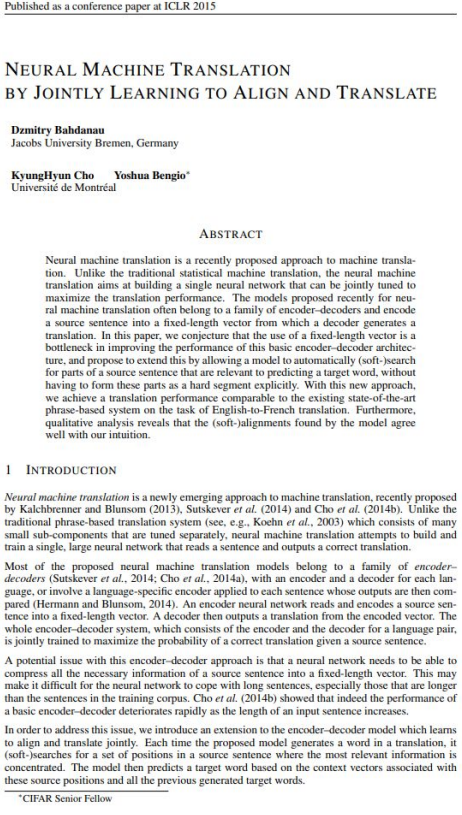
Computing an attention distribution assigns weights to input parts, determining their focus for generating the output

Additive Attention (Bahdanau et al.)
Dot-Product Attention (Luong et al.)
Scaled Dot-Product Attention (Vaswani et al., Transformer)

2 - Calculation of the context vector

The second step computes the context vector c_i , summarizing relevant input information using the attention distribution

Bahdanau (Additive) Model Adoption



Published as a conference paper at ICLR 2015

NEURAL MACHINE TRANSLATION
BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

Dzmitry Bahdanau
Jacobs University Bremen, Germany

KyungHyun Cho **Yoshua Bengio***
Université de Montréal

ABSTRACT

Neural machine translation is a recently proposed approach to machine translation. Unlike the traditional statistical machine translation, the neural machine translation aims at building a single neural network that can be jointly tuned to maximize the translation performance. The models proposed recently for neural machine translation often belong to a family of encoder–decoders and encode a source sentence into a fixed-length vector from which a decoder generates a translation. In this paper, we conjecture that the use of a fixed-length vector is a bottleneck in improving the performance of this basic encoder–decoder architecture, and propose to extend this by allowing a model to automatically (soft-)search for parts of a source sentence that are relevant to predicting a target word, without having to form these parts as a hard segment explicitly. With this new approach, we achieve a translation performance comparable to the existing state-of-the-art phrase-based system on the task of English-to-French translation. Furthermore, qualitative analysis reveals that the (soft-)alignments found by the model agree well with our intuition.

1 INTRODUCTION

Neural machine translation is a newly emerging approach to machine translation, recently proposed by Kalchbrenner and Blunsom (2013), Sutskever *et al.* (2014) and Cho *et al.* (2014b). Unlike the traditional phrase-based translation system (see, e.g., Koehn *et al.*, 2003) which consists of many small sub-components that are tuned separately, neural machine translation attempts to build and train a single, large neural network that reads a sentence and outputs a correct translation.

Most of the proposed neural machine translation models belong to a family of *encoder–decoders* (Sutskever *et al.*, 2014; Cho *et al.*, 2014a), with an encoder and a decoder for each language, or involve a language-specific encoder applied to each sentence whose outputs are then compared (Hermann and Blunsom, 2014). An encoder neural network reads and encodes a source sentence into a fixed-length vector. A decoder then outputs a translation from the encoded vector. The whole encoder–decoder system, which consists of the encoder and the decoder for a language pair, is jointly trained to maximize the probability of a correct translation given a source sentence.

A potential issue with this encoder–decoder approach is that a neural network needs to be able to compress all the necessary information of a source sentence into a fixed-length vector. This may make it difficult for the neural network to cope with long sentences, especially those that are longer than the sentences in the training corpus. Cho *et al.* (2014b) showed that indeed the performance of a basic encoder–decoder deteriorates rapidly as the length of an input sentence increases.

In order to address this issue, we introduce an extension to the encoder–decoder model which learns to align and translate jointly. Each time the proposed model generates a word in a translation, it (soft-)searches for a set of positions in a source sentence where the most relevant information is concentrated. The model then predicts a target word based on the context vectors associated with these source positions and all the previous generated target words.

*CIFAR Senior Fellow

1

However, researchers introduced variations to improve efficiency, scalability, or performance

Global vs. Local Attention

Self-Attention

Multi-Head Attention

Hard vs. Soft Attention

Key (K), Query(Q) & Value Functions

Keys

Keys (K) are derived from the **source data** (e.g., encoder hidden states, image features, or text embeddings)

They represent the elements of the input sequence or data that the model can attend to

For example:

In machine translation, ***K*** could be the hidden states of the encoder (e.g., ***hj*** in Bahdanau attention)
In image captioning, *K* could be feature vectors extracted from different regions of an image.
In document summarization, ***K*** could be embeddings of sentences or paragraphs.

WHAT INFORMATION DO WE HAVE?

["cat", "apple", "tree", "juice"] (representations of words in English)

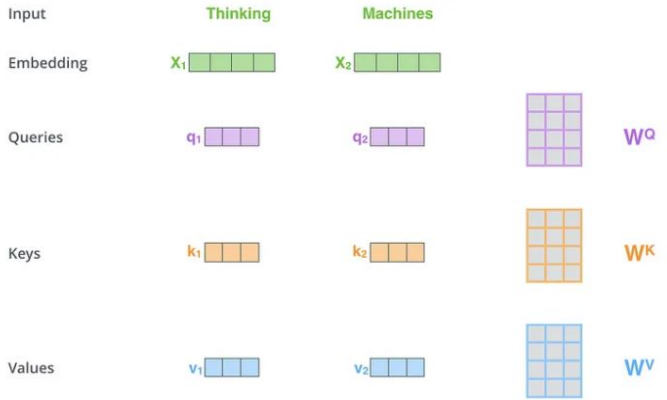


Figure 14. Key, Query and Value Functions
https://rahulrajpvr7d.medium.com/what-are-the-query-key-and-value-vectors-5656b8ca5fa0

Key (K), Query(Q) & Value Functions

Query & Value

The Query represents the current focus of the model, often derived from the decoder's hidden state (e.g., s_{t-1} in RNN-based models)

It is used to **compute the relevance or compatibility with each Key (K)** in the source data

For example:

In machine translation, ***Q*** could be the previous hidden states of the decoder (s_{t-1})
In Transformer models, *Q* is a learned representation of the current position in the output sequence

WHAT INFORMATION DO WE HAVE NEED?

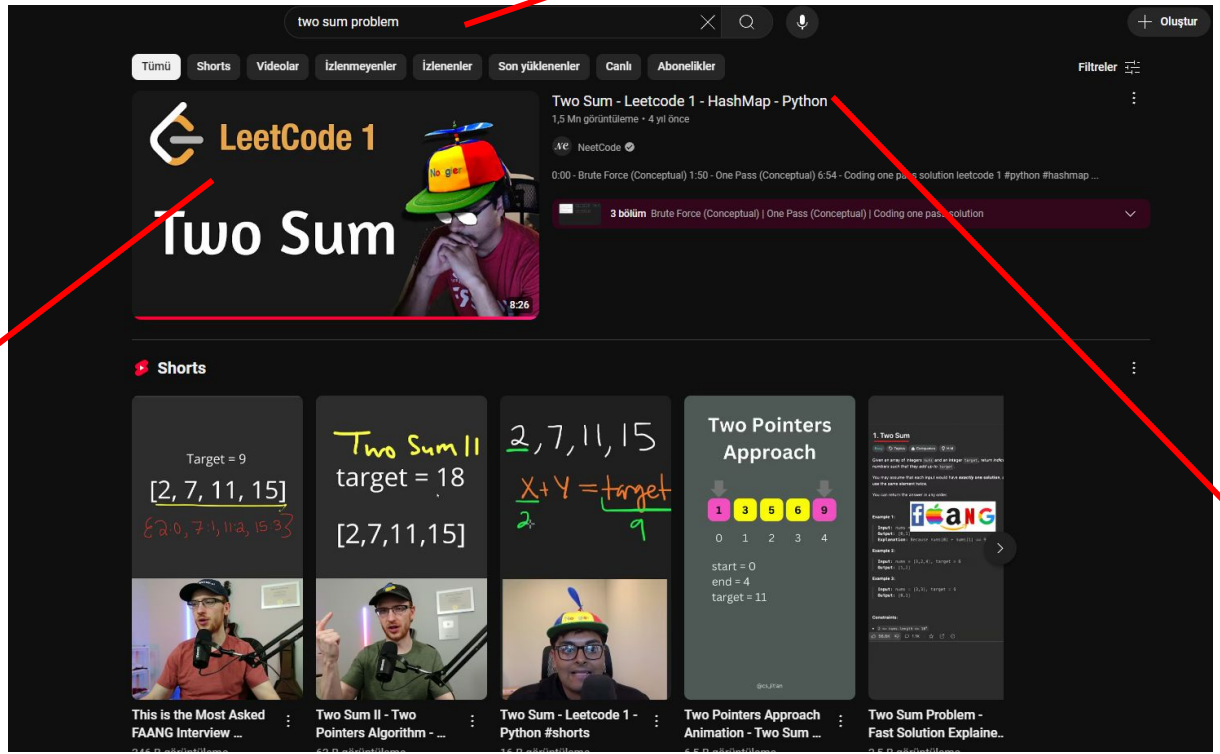
Values (V) represent the **actual information** or features **that the model extracts from the source data**
They are often derived from the same source as the Keys (K) but serve a different purpose

WHAT INFORMATION WILL BE BROADCASTED?

Key (K), Query(Q) & Value Functions

Real Life Example, Youtube

Query (Q)
“what information do we need? “



Value (V)
“what information do we broadcast? “

Key (K)
“what information do we have? “

Attention Mechanism

The goal is to determine the relationship (weights) between Q and all K's using a scoring function f (energy/compatibility function)

This calculates **energy scores**, **indicating Q's importance relative to all Ks, before generating the next output**

$$\boldsymbol{e} = f(\boldsymbol{q}, \boldsymbol{K}).$$

Name	Equation
Additive	$f(q,k) = v^{\top} act(W_1 k + W_2 q + b)$
Multiplicative (dot-product)	$f(q,k) = q^{\top} k$
Scaled multiplicative	$f(q,k) = \frac{q^{\top} k}{\sqrt{d_k}}$
General	$f(q,k) = q^{\top} W k$
Concat	$f(q,k) = v^{\top} act(W[k;q] + b)$
Location-based	$f(q,k) = f(q)$
Similarity	$f(q,k) = \frac{q \cdot k}{\|\|q\|\| \cdot \|\|k\|\|}$

← 2015 Paper

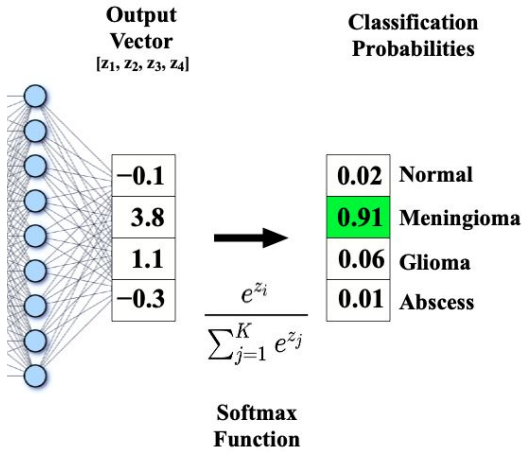
Table 1. Learnable parameters are k, v, b, W, W2, W3; d_k is the dimension of the input vector (keys); the act is a nonlinear activation function (ie. tanh and ReLU). Table by author.

Attention Mechanism

After that energy scores are fed into an attention distribution function named g, like the softmax layer in the RNN-search to compute the attention weights

$$\boldsymbol{\alpha} = g(\boldsymbol{e}).$$

by normalizing all energy scores to a probability distribution



There can be some drawbacks using Softmax

Softmax assigns a non-zero probability to every element in the input vector z, **even if some elements are irrelevant or have very low scores**

This can lead to **computational overhead**

Figure 14. Softmax Function
source : https://www.singlestore.com/blog/a-guide-to-softmax-activation-function/

Attention Mechanism Continued

- Using keys for both context vectors and attention distributions complicates training.
- Instead, it's more effective to introduce a separate feature vector, V, to explicitly differentiate the representations.
- In key-value attention, K and V represent different aspects of the same input data, and in **self-attention**, K, Q, and V are **distinct embeddings derived from the input**

$$\boldsymbol{c} = \phi(\{\alpha_i\}, \{\boldsymbol{v}_i\}),$$

where the ϕ is usually a weighted sum of V and is represented as a single vector:

$$\boldsymbol{z}_i = \alpha_i \boldsymbol{v}_i,$$

and,

$$\boldsymbol{c} = \sum_{i=1}^{n} \boldsymbol{z}_i,$$

where Zi is a weighted representation of V elements and n represents the size of Z.

Attention Mechanism Summary

Common attention mechanisms

“can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.”

as expressed by Vaswani et al. [7].

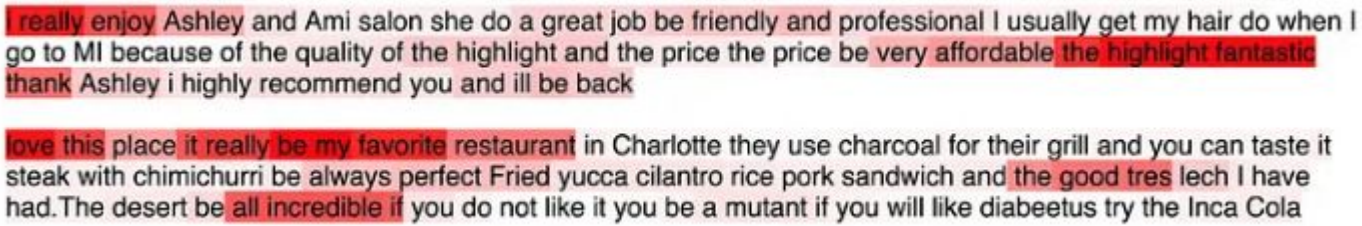


Figure X. Softmax Function
source : https://www.singlestore.com/blog/a-guide-to-softmax-activation-function/

Attention Mechanism Example

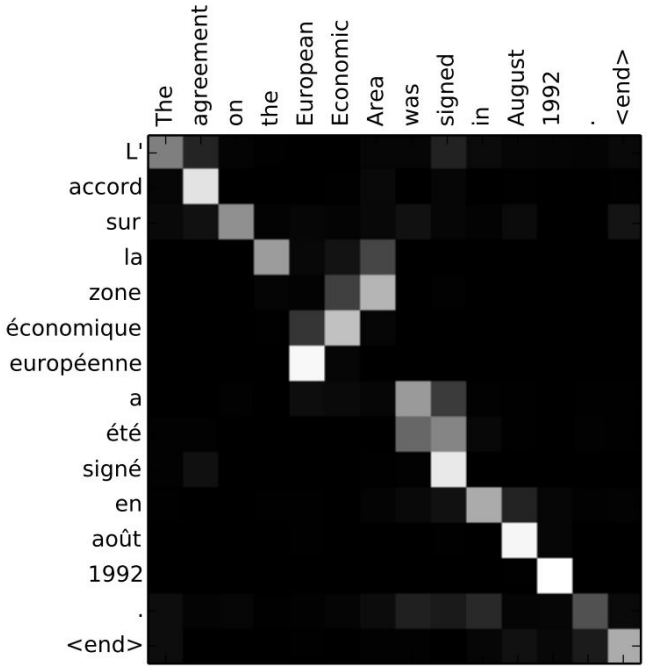
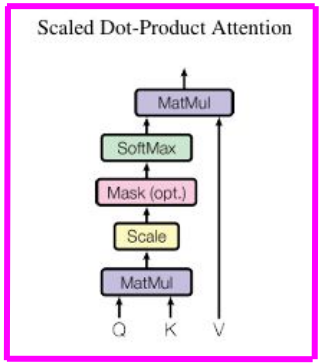
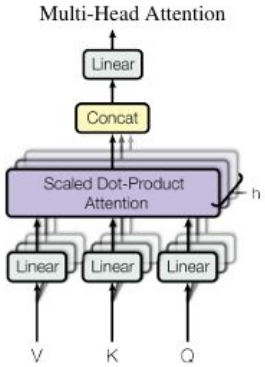


Figure X. Attention Amongst Words
source :https://paperswithcode.com/method/additive-attention

Self- Attention





Scaled Dot-Product Attention is the fundamental building block of **many attention mechanisms**

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

*Equal contribution. Listing order is random. Jakob proposed replacing RNNs with self-attention and started the effort to evaluate this idea. Ashish, with Illia, designed and implemented the first Transformer models and has been crucially involved in every aspect of this work. Noam proposed scaled dot-product attention, multi-head attention and the parameter-free position representation and became the other person involved in nearly every detail. Niki designed, implemented, tuned and evaluated countless model variants in our original codebase and tensor2tensor. Llion also experimented with novel model variants, was responsible for our initial codebase, and efficient inference and visualizations. Lukasz and Aidan spent countless long days designing various parts of and implementing tensor2tensor, replacing our earlier codebase, greatly improving results and massively accelerating our research.

†Work performed while at Google Brain.

‡Work performed while at Google Research.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

Self-Attention

Step-by-step computation

1. Transform Input to Q, K, V (Linear Projection)

$$Q = XW_q, \quad K = XW_k, \quad V = XW_v$$

2. Compute Attention Scores (Dot Product of Q and K)

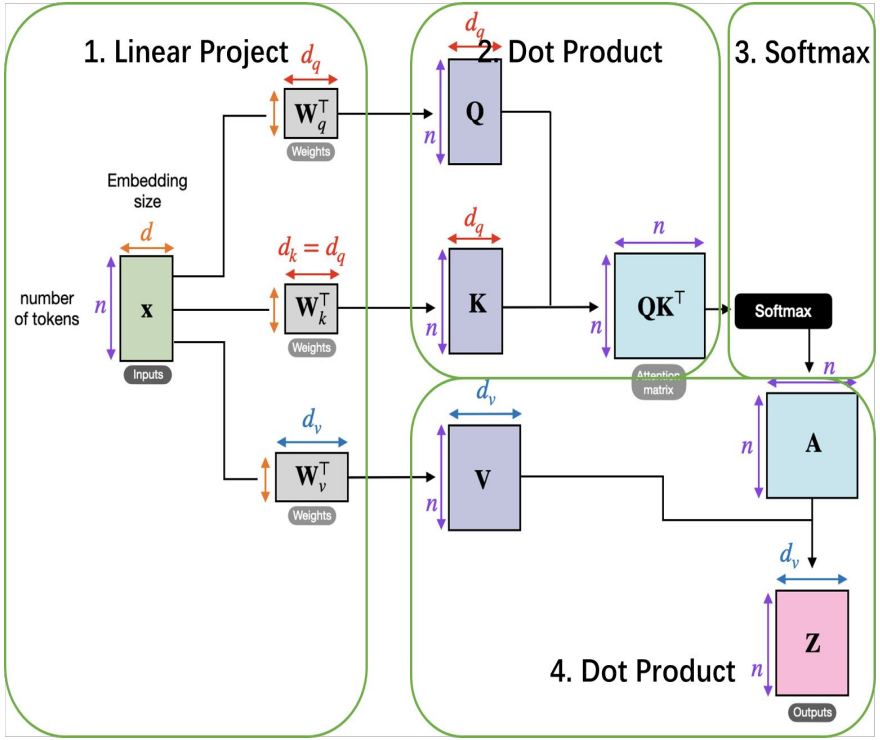
$$\text{Scaled Scores} = \frac{QK^T}{\sqrt{d_k}}$$

3. Apply Softmax (Normalize Attention Scores)

$$\text{Attention Weights} = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$$

4. Compute Weighted Sum with V (Final Output Z)

$$\mathbf{Z} = \text{Attention Weights} \cdot V$$



reference article : https://sebastianraschka.com/blog/2023/self-attention-from-scratch.html

Intuition about Attention Application

1. Multi-Headed Attention

– Improves diversity & parallelization.

2. Masked Attention

– Prevents information leakage in autoregressive models.

3. Cross Attention

– Facilitates encoder and decoder interaction

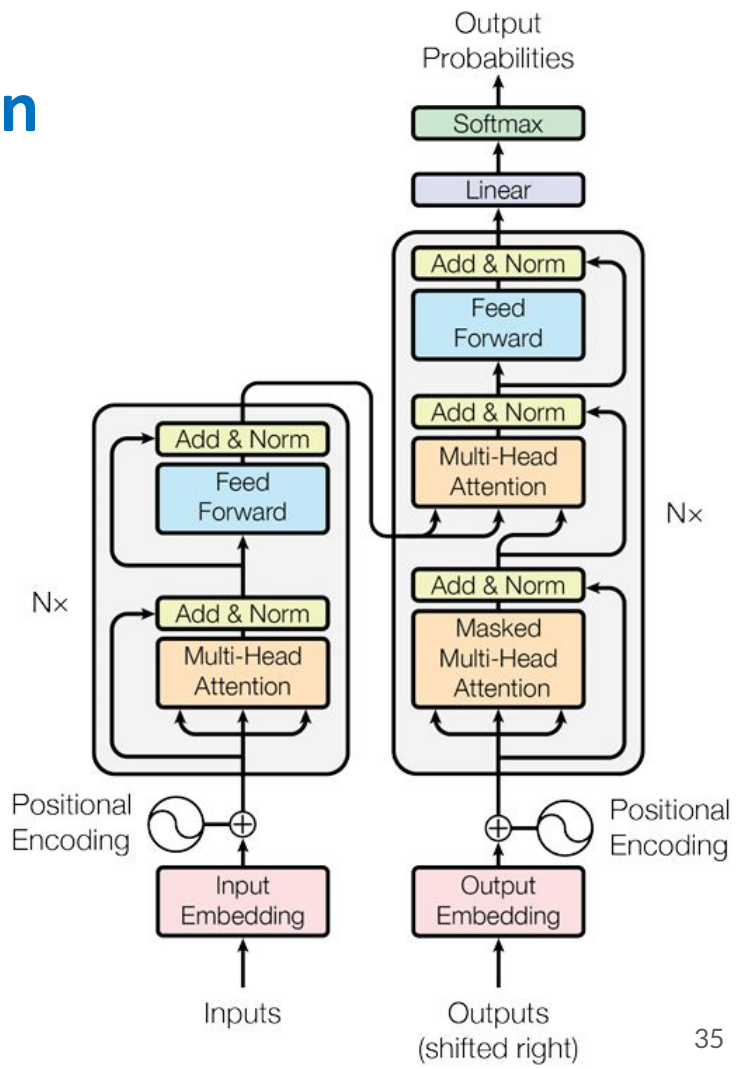
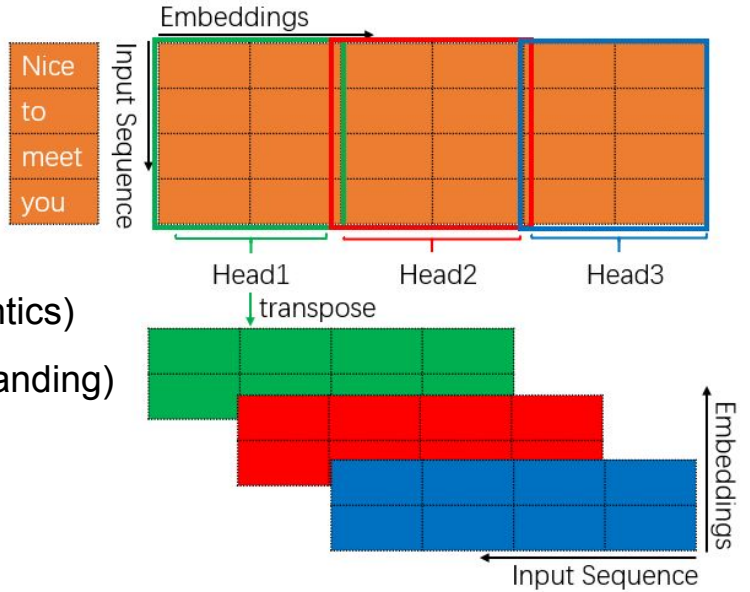


Figure: The Transformer- model architecture.
source : https://arxiv.org/abs/1706.03762

Multi-Headed Attention - Why Multiple Heads

Problem with Single-Head Attention:

Each head can focus on only **one type of** relationship



Benefit for Multi-Head Attention (MHSA):

Encodes different relationships (local, global, syntax, semantics)

Enhances representation learning (improves model understanding)

Enables parallelization (each head runs independently)

Used in NLP & Vision (BERT, ViT, GPT)

Equation

$$\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1,...,\text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Multi-Headed Attention - How it work

Step-by-step computation

1. Compute Q, K, and V using linear projections
2. Split Q, K, and V into multiple heads along Embeddings
3. Compute attention on each head independently
4. Concatenate attention from all heads together
5. Pass the final attention score through a FC layer

Equation

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

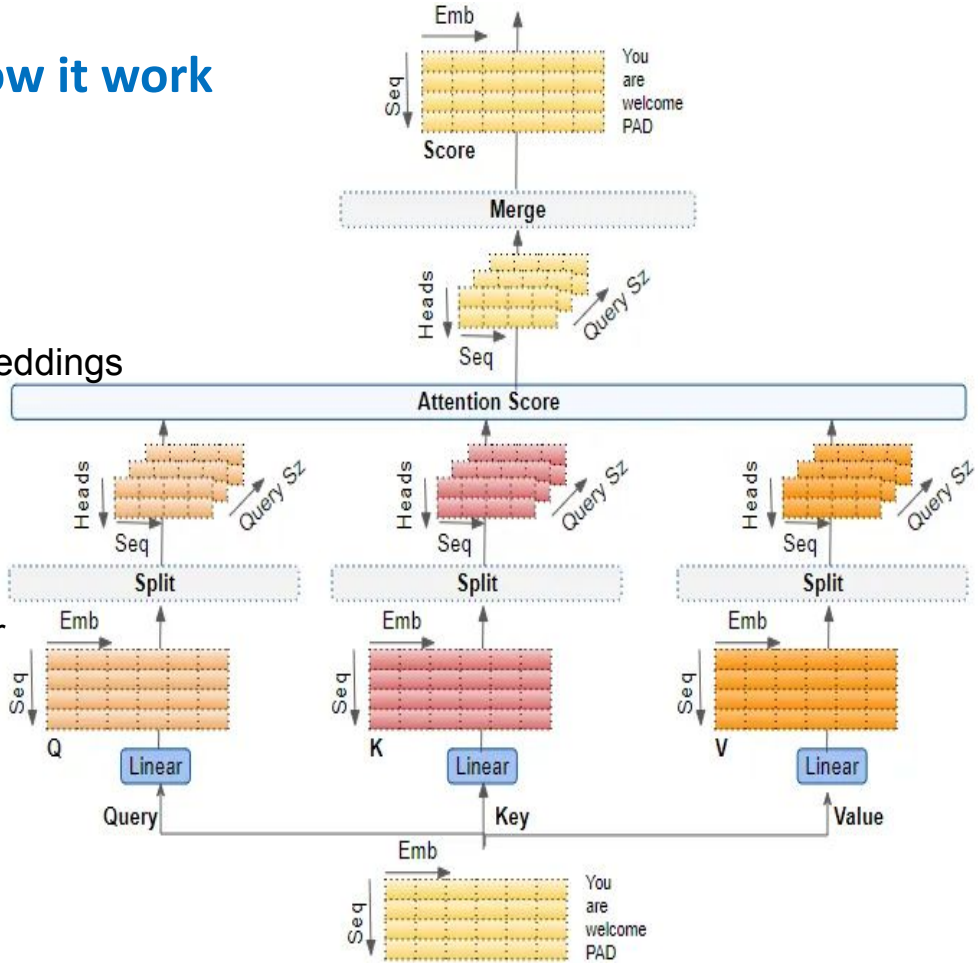


Figure: End-to-end flow of the Multi-head Attention
source : Transformers Explained Visually (Part 3): Multi-head Attention, deep dive | by Ketan Doshi | Towards Data Science

Masked Attention - Why Masked

Problem with Standard Attention:

Standard self-attention allows all tokens to attend to each other, leading to future information leakage

Proposed Method:

Masked Attention **blocks future tokens** to ensures each token **only attends to previous tokens**, enforcing a causal structure.

Equation

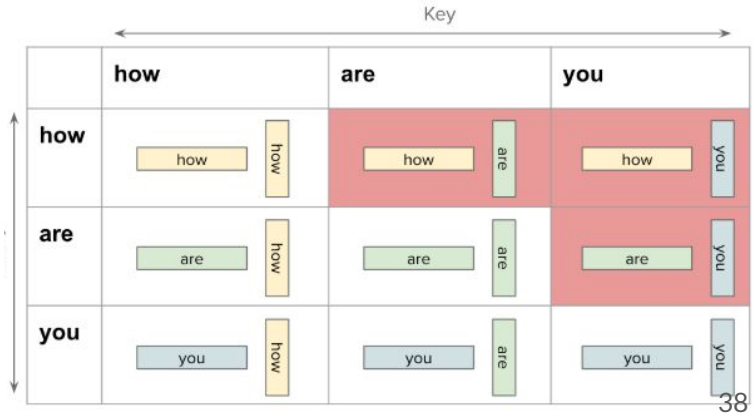
$$\text{Masked Scores} = QK^T + M$$

where M is the mask matrix, setting future token positions to $-\infty$.

Attention Mask

	how	are	you
how	0	-inf	-inf
are	0	0	-inf
you	0	0	0

Query/Key Dot Product



Masked Attention - How it work

Principal Intuition

Two tokens are highly similar, if the dot product is high.

Two tokens are unrelated to each other, if the dot product is negative infinity.

Step-by-step computation

1. The attention score matrix is computed as QK^T
2. A masking matrix is added to block future token positions.
3. The masked scores are passed through softmax and used to weight the values V

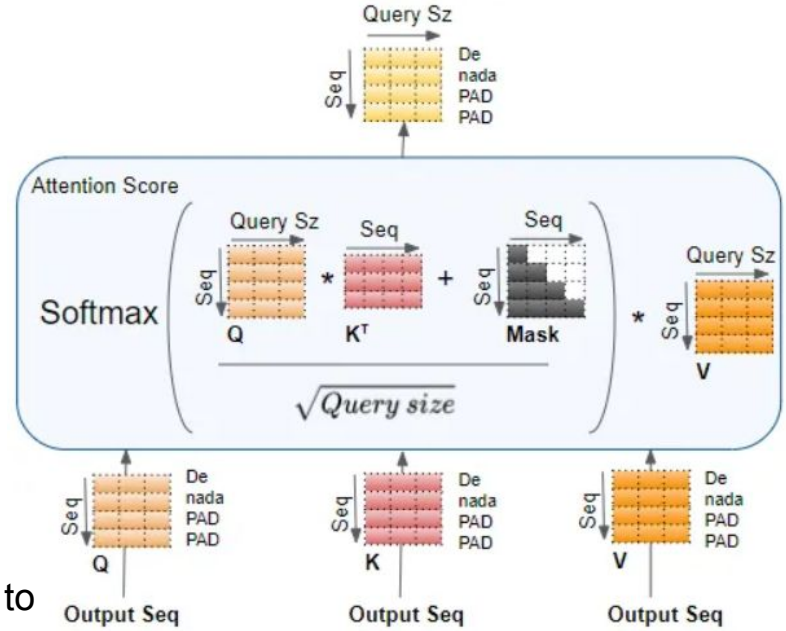


Figure: End-to-end flow of the Multi-head Attention

source : Transformers Explained Visually (Part 3): Multi-head Attention, deep dive | by Ketan Doshi | Towards Data Science

Cross Attention - Why Cross & How it works

What is Cross Attention?

Cross Attention is a mechanism that allow the decoder **attends to the encoder output** at each decoding step.

Step-by-step computation

1. Encoder provide **same Key and Value** vectors as it used in multi-head self-attention in Encoder
2. Decoder **compute new Query** by passing the output from masked-attention through a linear layer
3. Cross-attention layer in the decoder attends to the encoder outputs using Q,K,V

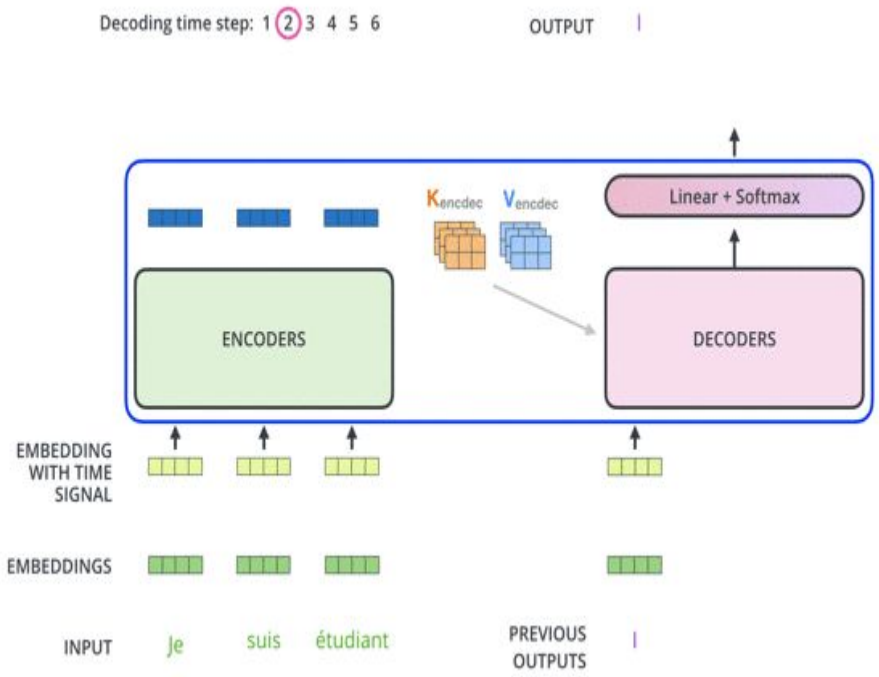


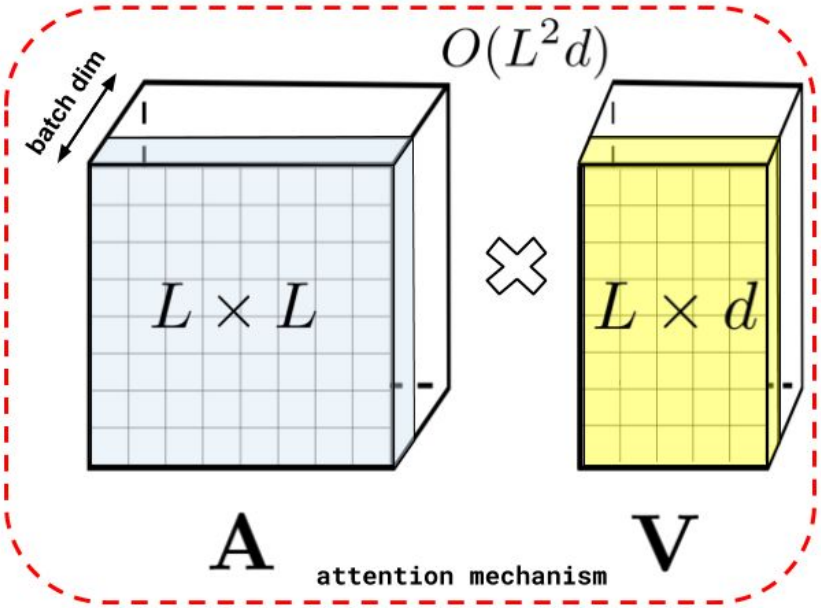
Figure: The Transformer- model architecture.
source : https://arxiv.org/abs/1706.03762

Challenges with Attention

Time Complexity:

$$O(L^2 d)$$

- Scaled Dot-Product Attention grows quadratically with sequence length.
- Complexity grows with each attention head.
- Computation occurs for every token repeatedly, thus we optimize by caching..



Where Q, K, V $\in R^{L \times d}$
L is the sequence length (number of tokens).
d is the hidden dimension (latent space size).

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

Attention in numbers

Llama3 70B LLM

- L = 80 layers
- H = 64 attention heads
- B = 8 batch size of 8 sequences
- dk = 128 key/value dimension
- 16-bit precision

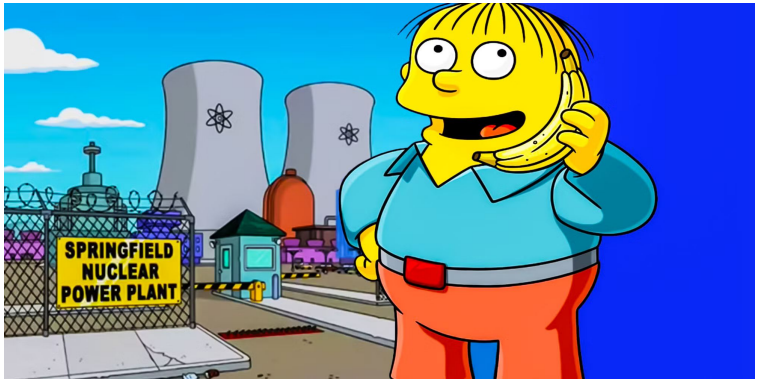
$$L \times H \times B \times n \times d_k \times 2 \times 2 \text{ bytes} = 80 \times 64 \times 8 \times 1000 \times 128 \times 2 \times 2 \text{ bytes} =$$

20.97GB Cache Size

References
https://ai.meta.com/blog/meta-llama-3/
https://www.pyspur.dev/blog/multi-head-latent-attention-kv-cache-paper-list

Attention is Expen$ive.

- Energy consumption is dominated by the attention mechanism.
- With billion+ parameter models only growing, this is not sustainable - we can't all build power plants.
- A more efficient and effective attention mechanism translates to faster training and inference.
- Lastly, lower compute costs would directly translate to CO_2 emission reduction and lower energy consumption.
- Better for the model, better for the environment.



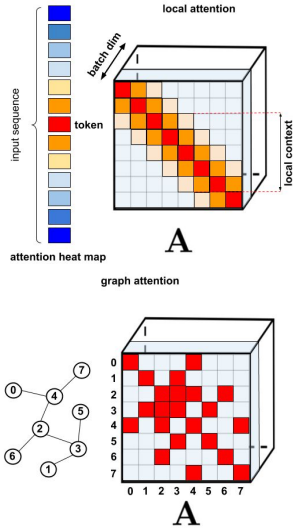
Meta and Microsoft's Solution to Computational Bottlenecks

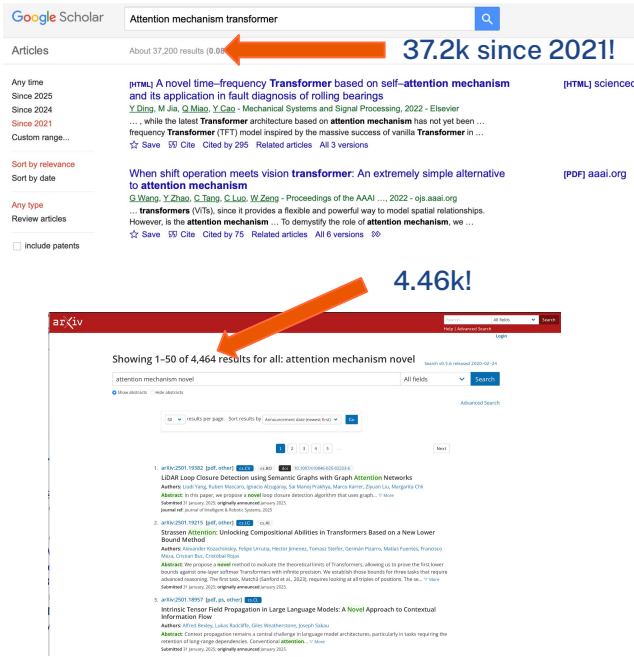
Attention Research

The challenges of the attention mechanism has made it a key topic in ML. Thus, this area has seen an explosion of research within the past few years.

Select methods that have been published:

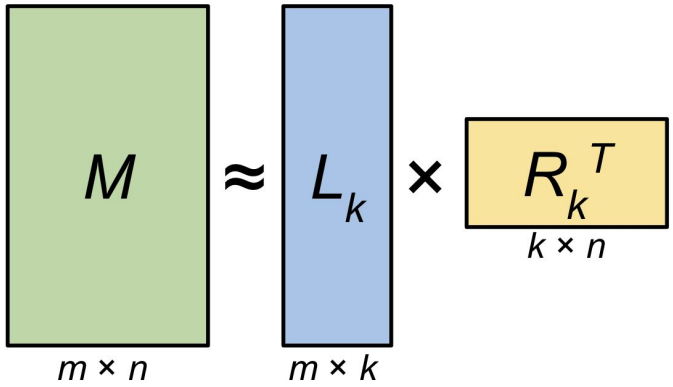
- Sliding Window
- Local Neighborhoods
- Graph Based
- Clustering & Binning
- Sparse Methods
- Linear Methods
- Low rank compression

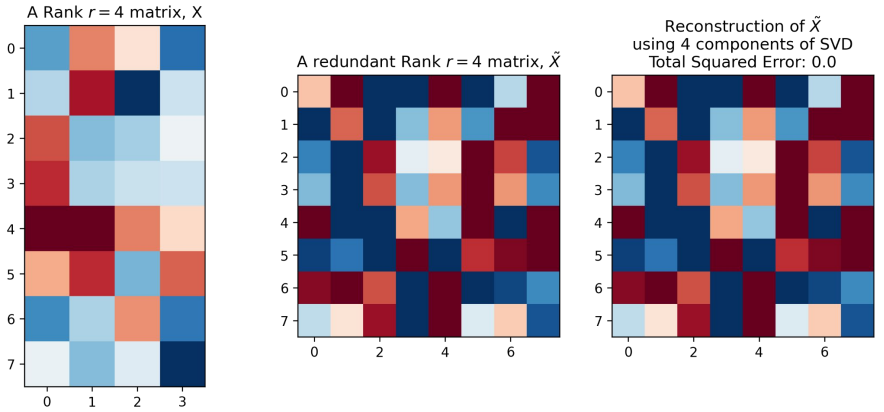




Low Rank Compression & Approximation Methods

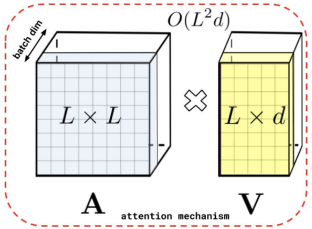
- Low Rank matrices has become a popular proposition.
- Rank refers to the number of independent rows or columns in a matrix.
- Low-rank matrices inherently contain redundant information, allowing for compression without significant loss of critical data.
- By transforming a matrix into a low-rank representation (e.g., using techniques like Singular Value Decomposition (SVD)), we effectively reduce the storage and computational requirements.

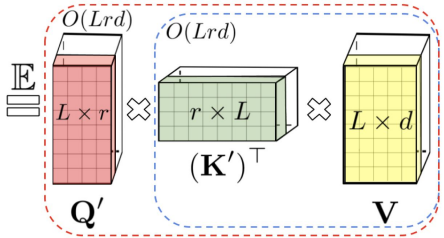




Performers
Fast Attention via Positive Orthogonal Random Features

- Google DeepMind Team in 2020.
- Uses kernel methods to approximate softmax.
- Reduces memory usage and improves efficiency.
- Avoids storing large key-value pairs.
- Helps scale attention for longer sequences.





October 2020

Published as a conference paper at ICLR 2021

RETHINKING ATTENTION WITH PERFORMERS

Krzysztof Choromanski[*1], **Valerii Likhosherstov**[*2], **David Dohan**[*1], **Xingyou Song**[*1]
Andreea Gane[*1], **Tamas Sarlos**[*1], **Peter Hawkins**[*1], **Jared Davis**[*3], **Afroz Mohiuddin**[1]
Lukasz Kaiser[1], **David Belanger**[1], **Lucy Colwell**[1,2], **Adrian Weller**[2,4]
[1]Google [2]University of Cambridge [3]DeepMind [4]Alan Turing Institute

ABSTRACT

We introduce *Performers*, Transformer architectures which can estimate regular (softmax) full-rank-attention Transformers with provable accuracy, but using only linear (as opposed to quadratic) space and time complexity, without relying on any priors such as sparsity or low-rankness. To approximate softmax attention-kernels, Performers use a novel *Fast Attention Via positive Orthogonal Random features* approach (FAVOR+), which may be of independent interest for scalable kernel methods. FAVOR+ can also be used to efficiently model kernelizable attention mechanisms beyond softmax. This representational power is crucial to accurately compare softmax with other kernels for the first time on large-scale tasks, beyond the reach of regular Transformers, and investigate optimal attention-kernels. Performers are linear architectures fully compatible with regular Transformers and with strong theoretical guarantees: unbiased or nearly-unbiased estimation of the attention matrix, uniform convergence and low estimation variance. We tested Performers on a rich set of tasks stretching from pixel-prediction through text models to protein sequence modeling. We demonstrate competitive results with other examined efficient sparse and dense attention methods, showcasing effectiveness of the novel attention-learning paradigm leveraged by Performers.

1 INTRODUCTION AND RELATED WORK

Transformers (Vaswani et al., 2017; Dehghani et al., 2019) are powerful neural network architectures that have become SOTA in several areas of machine learning including natural language processing (NLP) (e.g. speech recognition (Luo et al., 2020)), neural machine translation (NMT) (Chen et al., 2018), document generation/summarization, time series prediction, generative modeling (e.g. image generation (Parmar et al., 2018)), music generation (Huang et al., 2019), and bioinformatics (Rives et al., 2019; Madani et al., 2020; Ingraham et al., 2019; Elnaggar et al., 2019; Du et al., 2020).

Transformers rely on a trainable *attention* mechanism that identifies complex dependencies between the elements of each input sequence. Unfortunately, the regular Transformer scales quadratically with the number of tokens L in the input sequence, which is prohibitively expensive for large L and precludes its usage in settings with limited computational resources even for moderate values of L. Several solutions have been proposed to address this issue (Beltagy et al., 2020; Gulati et al., 2020; Chan et al., 2020; Child et al., 2019; Bello et al., 2019). Most approaches restrict the attention mechanism to attend to local neighborhoods (Parmar et al., 2018) or incorporate structural priors on attention such as sparsity (Child et al., 2019), pooling-based compression (Rae et al., 2020) clustering/binning/convolution techniques (e.g. (Roy et al., 2020) which applies k-means clustering to learn dynamic sparse attention regions, or (Kitaev et al., 2020), where locality sensitive hashing is used to group together tokens of similar embeddings), sliding windows (Beltagy et al., 2020), or truncated targeting (Chelba et al., 2020). There is also a long line of research on using dense attention matrices, but defined by low-rank kernels substituting softmax (Katharopoulos et al., 2020; Shen et al., 2018). Those methods critically rely on kernels admitting explicit representations as dot-products of finite positive-feature vectors.

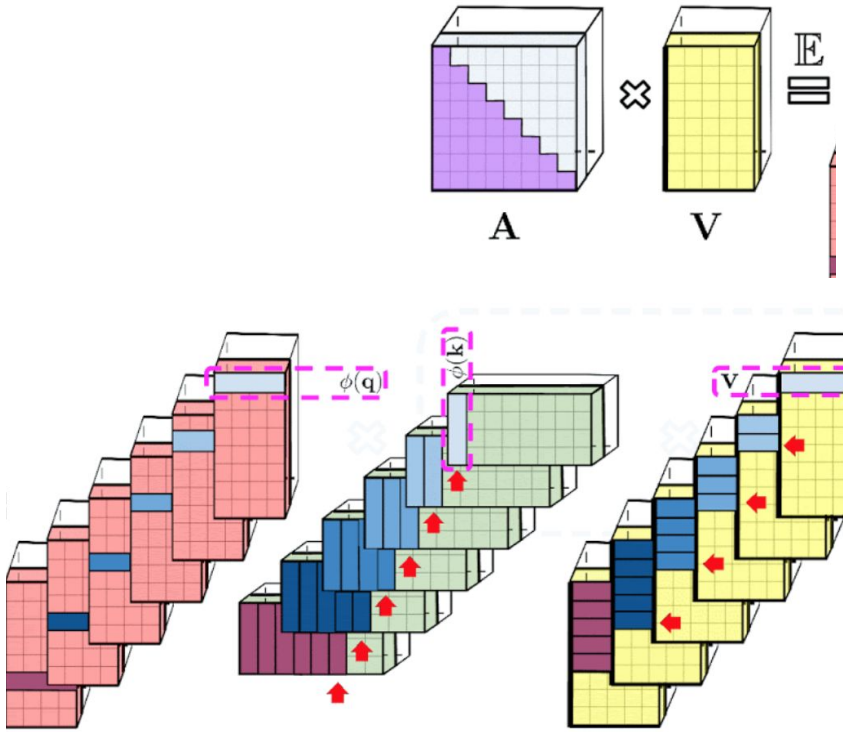
The approaches above do not aim to approximate regular attention, but rather propose simpler and more tractable attention mechanisms, often by incorporating additional constraints (e.g. identical query and key sets as in (Kitaev et al., 2020)), or by trading regular with sparse attention using more

[*]Equal contribution. Correspondence to {kchoro,lcolwell}@google.com.
Code for Transformer models on protein data can be found in github.com/google-research/google-research/tree/master/protein_lm and Performer code can be found in github.com/google-research/google-research/tree/master/performer. Google AI Blog: https://ai.googleblog.com/2020/10/rethinking-attention-with-performers.html

https://arxiv.org/pdf/2009.14794
https://research.google/blog/rethinking-attention-with-performers/

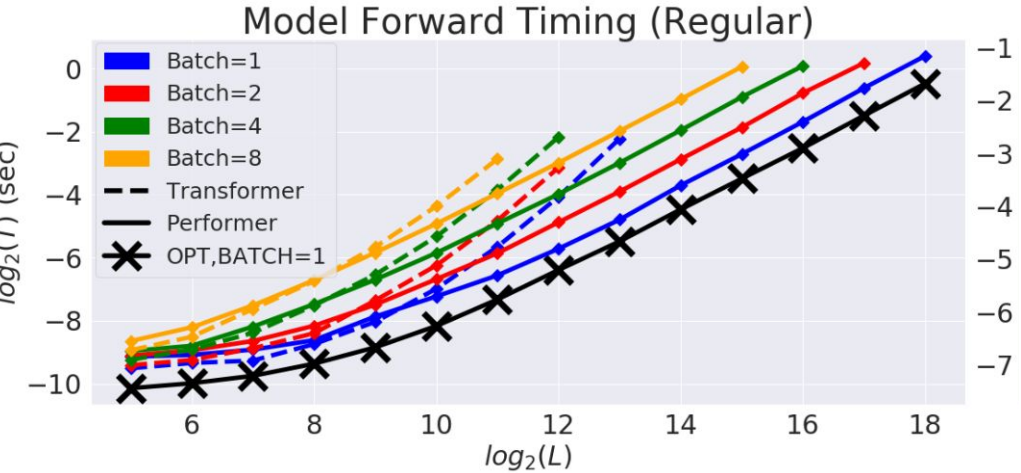
Performers

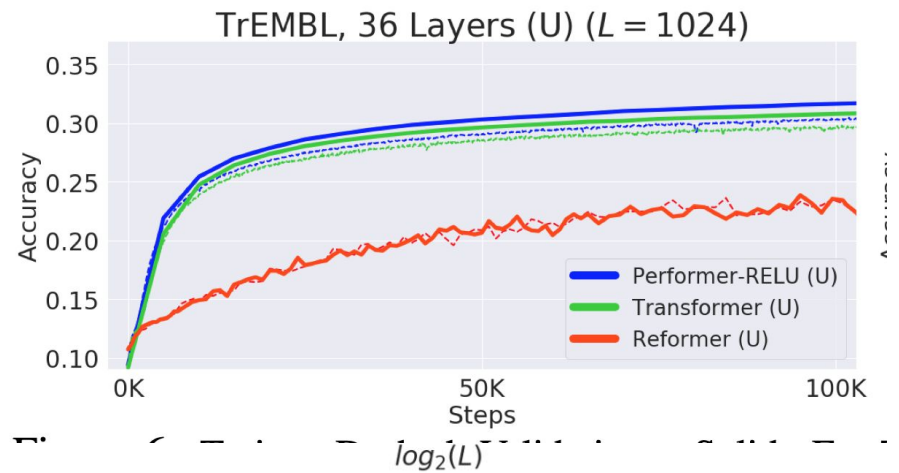
- Attention Matrix is decomposed to allow rearrangement and approximation



https://arxiv.org/pdf/2009.14794
https://research.google/blog/rethinking-attention-with-performers/

Performers Performance

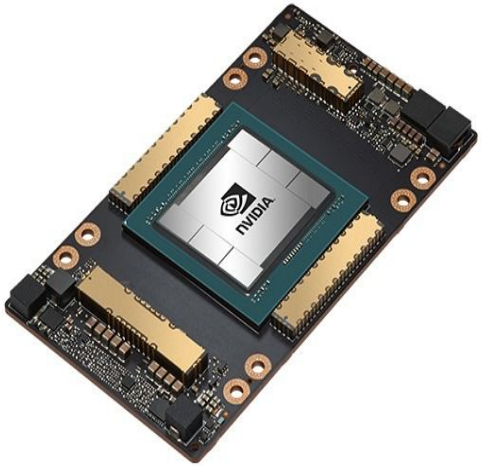




Flash Attention

June 2022

- Fast! Able to train BERT-large(seq512) 15% faster
- Memory Efficient with sub-quadratic time vs $O(N^2)$
- No approximation methods
- Hardware Aware



FLASHATTENTION: Fast and Memory-Efficient Exact Attention with IO-Awareness

Tri Dao[†], Daniel Y. Fu[†], Stefano Ermon[†], Atri Rudra[‡], and Christopher Ré[†]

[†]Department of Computer Science, Stanford University
[‡]Department of Computer Science and Engineering, University at Buffalo, SUNY
{trid,danfu}@cs.stanford.edu, ermon@stanford.edu, atri@buffalo.edu, chrismre@cs.stanford.edu

June 24, 2022

Abstract

Transformers are slow and memory-hungry on long sequences, since the time and memory complexity of self-attention are quadratic in sequence length. Approximate attention methods have attempted to address this problem by trading off model quality to reduce the compute complexity, but often do not achieve wall-clock speedup. We argue that a missing principle is making attention algorithms *IO-aware*—accounting for reads and writes between levels of GPU memory. We propose FLASHATTENTION, an IO-aware exact attention algorithm that uses tiling to reduce the number of memory reads/writes between GPU high bandwidth memory (HBM) and GPU on-chip SRAM. We analyze the IO complexity of FLASHATTENTION, showing that it requires fewer HBM accesses than standard attention, and is optimal for a range of SRAM sizes. We also extend FLASHATTENTION to block-sparse attention, yielding an approximate attention algorithm that is faster than any existing approximate attention method. FLASHATTENTION trains Transformers faster than existing baselines: 15% end-to-end wall-clock speedup on BERT-large (seq. length 512) compared to the MLPerf 1.1 training speed record, 3× speedup on GPT-2 (seq. length 1K), and 2.4× speedup on long-range arena (seq. length 1K-4K). FLASHATTENTION and block-sparse FLASHATTENTION enable longer context in Transformers, yielding higher quality models (0.7 better perplexity on GPT-2 and 6.4 points of lift on long-document classification) and entirely new capabilities: the first Transformers to achieve better-than-chance performance on the Path-X challenge (seq. length 16K, 61.4% accuracy) and Path-256 (seq. length 64K, 63.1% accuracy).

1 Introduction

Transformer models [82] have emerged as the most widely used architecture in applications such as natural language processing and image classification. Transformers have grown larger [5] and deeper [83], but equipping them with longer context remains difficult [80], since the self-attention module at their heart has time and memory complexity quadratic in sequence length. An important question is whether making attention faster and more memory-efficient can help Transformer models address their runtime and memory challenges for long sequences.

Many approximate attention methods have aimed to reduce the compute and memory requirements of attention. These methods range from sparse-approximation [51, 74] to low-rank approximation [12, 50, 84], and their combinations [3, 9, 92]. Although these methods reduce the compute requirements to linear or near-linear in sequence length, many of them do not display wall-clock speedup against standard attention and have not gained wide adoption. One main reason is that they focus on FLOP reduction (which may not correlate with wall-clock speed) and tend to ignore overheads from memory access (IO).

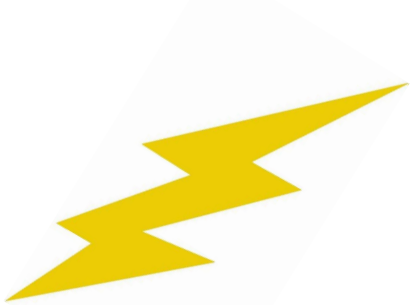
In this paper, we argue that a missing principle is making attention algorithms *IO-aware* [1]—that is, carefully accounting for reads and writes to different levels of fast and slow memory (e.g., between fast GPU on-chip SRAM and relatively slow GPU high bandwidth memory, or HBM [45], Figure 1 left). On modern

1

References

[2205.14135] FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness

Flash Attention
Compute & Memory Bottlenecks

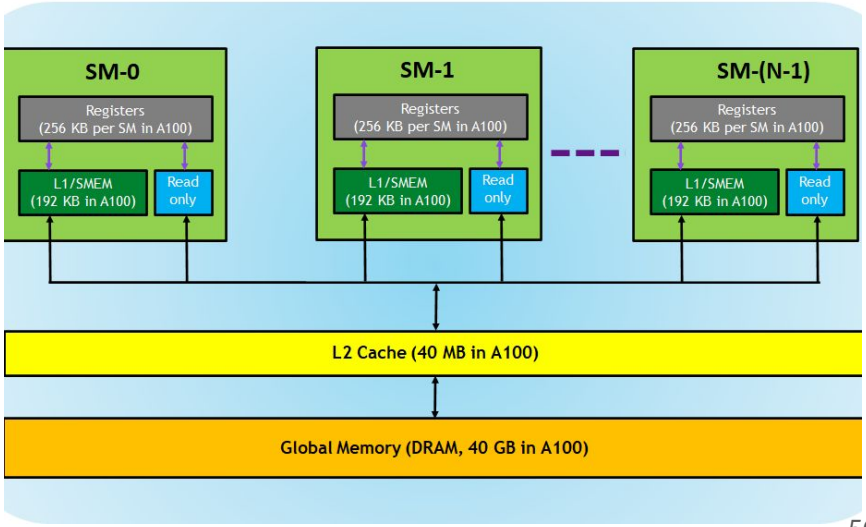


Compute

- GPU - performs the computations such as with matrix multiplication.
- Memory - stores data locally within the GPU for fast access.

This creates two potential **bottlenecks**: compute (limited processing power) and memory (bandwidth and capacity constraints).

Attention mechanisms are highly memory-bound, primarily due to key-value (KV) caching, which requires significant storage during inference.



References
https://developer.nvidia.com/blog/cuda-refresher-cuda-programming-model/

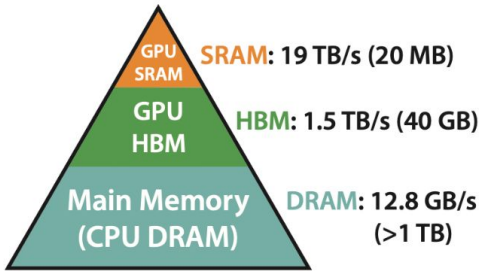
Flash Attention

Flash attention takes into account reads and writes to different levels of fast and slow memory:

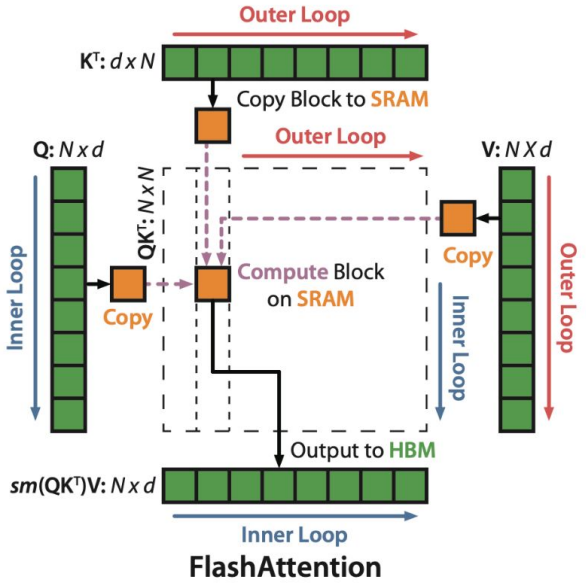
Fast GPU on-chip SRAM and relatively **slow GPU high bandwidth memory**).

NVIDIA'S A100: 40-80GB of High Bandwidth Memory (1.5-2TB/s)

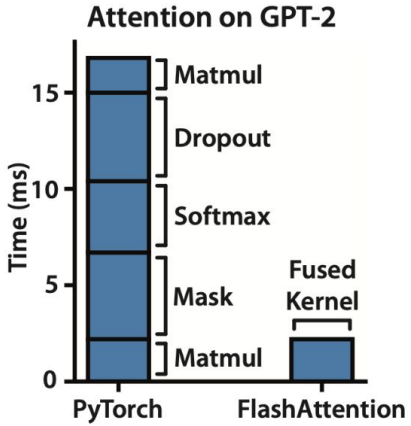
192kb of L1 Low Bandwidth SRAM x 108 Stream processors (19TB/s)



Memory Hierarchy with Bandwidth & Memory Size



FlashAttention



References
[2205.14135] FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness
https://developer.nvidia.com/blog/cuda-refresher-cuda-programming-model/

Flash Attention

Main Idea

Compute attention with least amount of HMB I/O

Method

Q,K,V is loaded from HBM and split into blocks (Tiling).

Blocks loaded to SRAM

Attention computed w.r.t current block.

Softmax.

Backward pass.

Write back and repeat for next block.

Algorithm 0 Standard Attention Implementation

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM.

1: Load $\mathbf{Q}, \mathbf{K}$ by blocks from HBM, compute $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top$, write $\mathbf{S}$ to HBM.
2: Read $\mathbf{S}$ from HBM, compute $\mathbf{P} = \mathrm{softmax}(\mathbf{S})$, write $\mathbf{P}$ to HBM.
3: Load $\mathbf{P}$ and $\mathbf{V}$ by blocks from HBM, compute $\mathbf{O} = \mathbf{P}\mathbf{V}$, write $\mathbf{O}$ to HBM.
4: Return $\mathbf{O}$.

Algorithm 1 FlashAttention

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM, on-chip SRAM of size M.

1: Set block sizes $B_c = \lceil \frac{M}{4d} \rceil$, $B_r = \min\left(\lceil \frac{M}{4d} \rceil, d\right)$.
2: Initialize $\mathbf{O} = (0)_{N \times d} \in \mathbb{R}^{N \times d}, \ell = (0)_N \in \mathbb{R}^N, m = (-\infty)_N \in \mathbb{R}^N$ in HBM.
3: Divide $\mathbf{Q}$ into $T_r = \left\lceil \frac{N}{B_r} \right\rceil$ blocks $\mathbf{Q}_1, \dots, \mathbf{Q}_{T_r}$ of size $B_r \times d$ each, and divide $\mathbf{K}, \mathbf{V}$ in to $T_c = \left\lceil \frac{N}{B_c} \right\rceil$ blocks $\mathbf{K}_1, \dots, \mathbf{K}_{T_c}$ and $\mathbf{V}_1, \dots, \mathbf{V}_{T_c}$, of size $B_c \times d$ each.
4: Divide $\mathbf{O}$ into T_r blocks $\mathbf{O}_i, \dots, \mathbf{O}_{T_r}$ of size $B_r \times d$ each, divide ℓ into T_r blocks $\ell_i, \dots, \ell_{T_r}$ of size B_r each, divide m into T_r blocks $m_1, \dots, m_{T_r}$ of size B_r each.
5: **for** $1 \le j \le T_c$ **do**
6: Load $\mathbf{K}_j, \mathbf{V}_j$ from HBM to on-chip SRAM.
7: **for** $1 \le i \le T_r$ **do**
8: Load $\mathbf{Q}_i, \mathbf{O}_i, \ell_i, m_i$ from HBM to on-chip SRAM.
9: On chip, compute $\mathbf{S}_{ij} = \mathbf{Q}_i \mathbf{K}_j^T \in \mathbb{R}^{B_r \times B_c}$.
10: On chip, compute $\tilde{m}_{ij} = \mathrm{rowmax}(\mathbf{S}_{ij}) \in \mathbb{R}^{B_r}$, $\tilde{\mathbf{P}}_{ij} = \exp(\mathbf{S}_{ij} - \tilde{m}_{ij}) \in \mathbb{R}^{B_r \times B_c}$ (pointwise), $\tilde{\ell}_{ij} = \mathrm{rowsum}(\tilde{\mathbf{P}}_{ij}) \in \mathbb{R}^{B_r}$.
11: On chip, compute $m_i^{\text{new}} = \max(m_i, \tilde{m}_{ij}) \in \mathbb{R}^{B_r}$, $\ell_i^{\text{new}} = e^{m_i - m_i^{\text{new}}} \ell_i + e^{\tilde{m}_{ij} - m_i^{\text{new}}} \tilde{\ell}_{ij} \in \mathbb{R}^{B_r}$.
12: Write $\mathbf{O}_i \leftarrow \mathrm{diag}(\ell_i^{\text{new}})^{-1}(\mathrm{diag}(\ell_i) e^{m_i - m_i^{\text{new}}} \mathbf{O}_i + e^{\tilde{m}_{ij} - m_i^{\text{new}}} \tilde{\mathbf{P}}_{ij} \mathbf{V}_j)$ to HBM.
13: Write $\ell_i \leftarrow \ell_i^{\text{new}}$, $m_i \leftarrow m_i^{\text{new}}$ to HBM.
14: **end for**
15: **end for**
16: Return $\mathbf{O}$.

References
[2205.14135] FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness

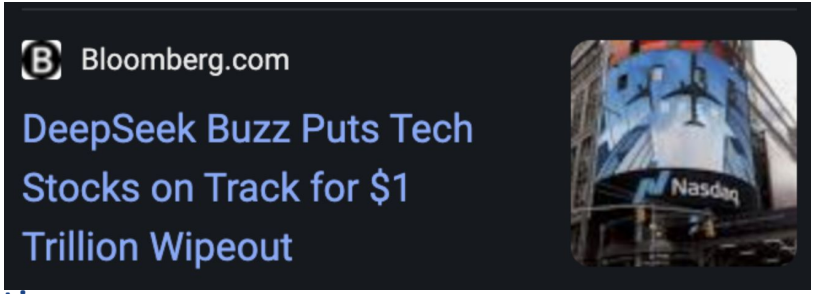
Flash Attention Performance

Model implementations	OpenWebText (ppl)	Training time (speedup)
GPT-2 small - Huggingface [87]	18.2	9.5 days (1.0×)
GPT-2 small - Megatron-LM [77]	18.2	4.7 days (2.0×)
GPT-2 small - FlashAttention	18.2	**2.7 days (3.5×)**
GPT-2 medium - Huggingface [87]	14.2	21.0 days (1.0×)
GPT-2 medium - Megatron-LM [77]	14.3	11.5 days (1.8×)
GPT-2 medium - FlashAttention	14.3	**6.9 days (3.0×)**

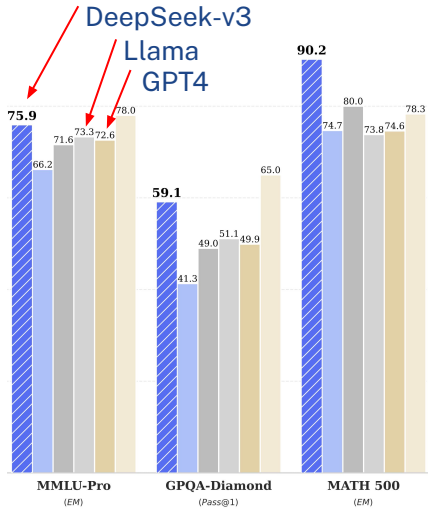
8x NVIDIA A100s

References
[2205.14135] FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness

Multi Head Latent Attention



- DeepSeek-v2 introduces Multi-Head Latent Attention (MLA) - May 2024
- DS-v3 uses MLA to achieve efficient inference & cost-effective training - December 2024



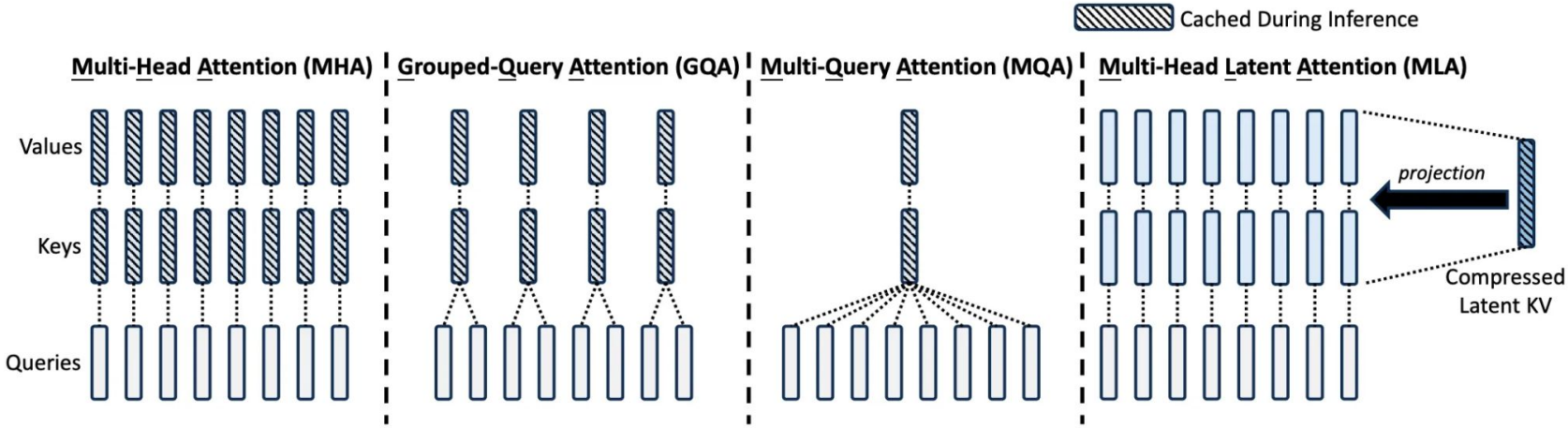
Abstract

We present DeepSeek-V3, a strong Mixture-of-Experts (MoE) language model with 671B total parameters with 37B activated for each token. To achieve efficient inference and cost-effective training, DeepSeek-V3 adopts Multi-head Latent Attention (MLA) and DeepSeekMoE architectures, which were thoroughly validated in DeepSeek-V2. Furthermore, DeepSeek-V3 pioneers an auxiliary-loss-free strategy for load balancing and sets a multi-token prediction training objective for stronger performance. We pre-train DeepSeek-V3 on 14.8 trillion diverse and high-quality tokens, followed by Supervised Fine-Tuning and Reinforcement Learning stages to

References:
[2405.04434] DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model

DeepSeek-V3 Technical Report

DeepSeek
Multi Head Latent Attention



Best in theory	**Medium Performance**	**Weaker Performance**	**Strong Performance**
KV Overhead intense	Significantly less KV Cache	Very small KV Cache	Small KV Cache
KV Cache per Token	**KV Cache per Token**	**KV Cache per Token**	**KV Cache per Token**
$2n_h d_h l$	$2n_g d_h l$	$2_g d_h l$	$(d_c+d^R_h)l = 9/2_g d_h l$

n_h is number of heads d_h is dimension per head l is number of layers n_g is number of subgroups d_c is compression dimension

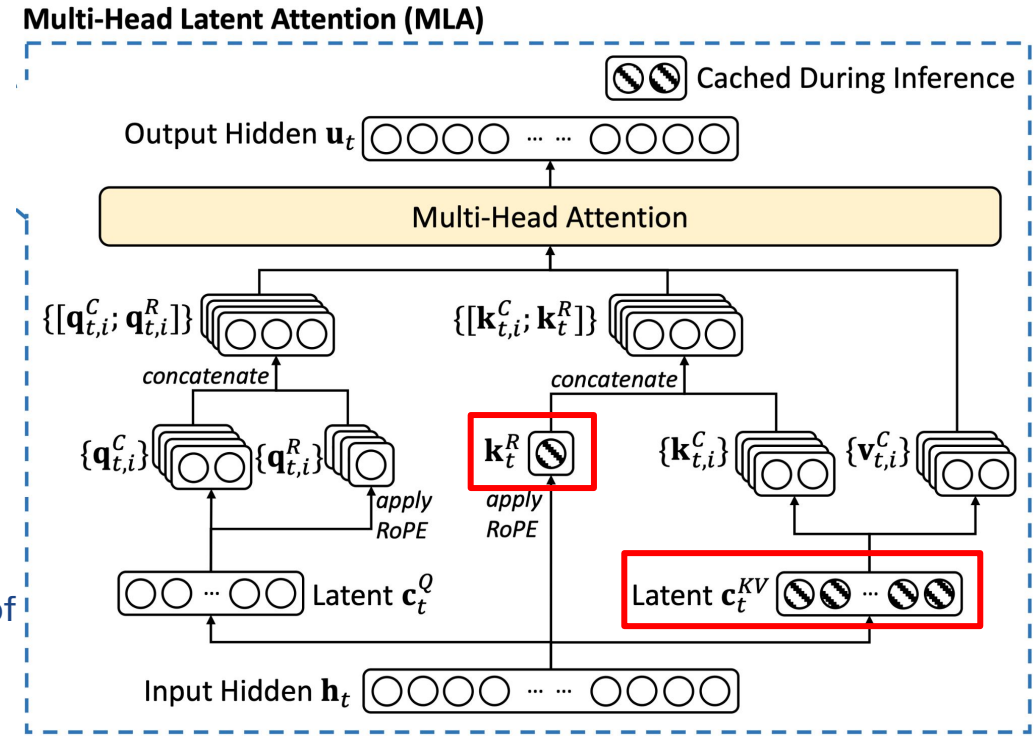
References:
[2405.04434] DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model
DeepSeek-V3 Technical Report

DeepSeek
Multi Head Latent Attention

- Down projects the KV into a low-rank, compressed latent space.
- During inference, compressed KVs are projected up

Where C^{KV}_t is the compressed low-D representation of the original KV vectors

Rotary Positional Embeddings (RoPE) is also part of the mechanism but omitted in this discussion.



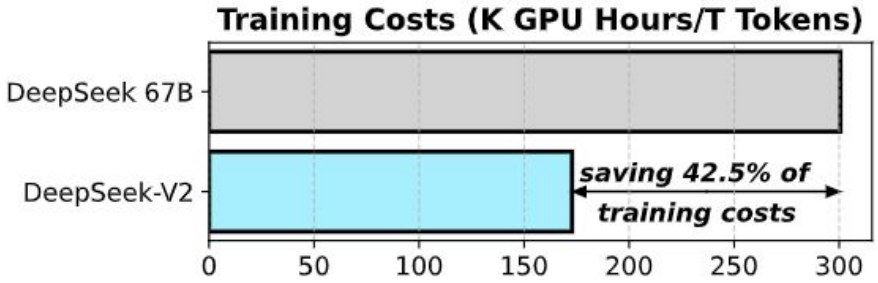
Only vectors in the red box need to be cached

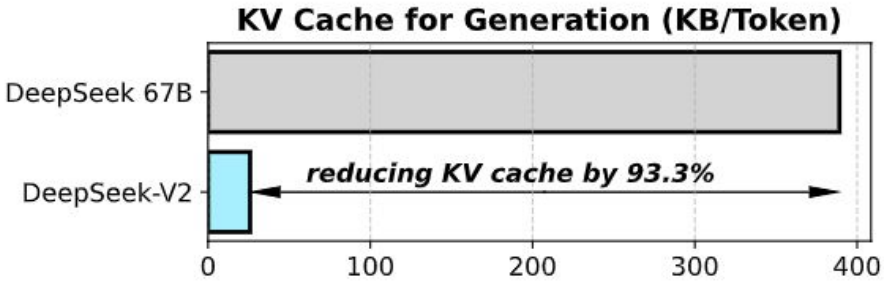
References:
[2405.04434] DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model

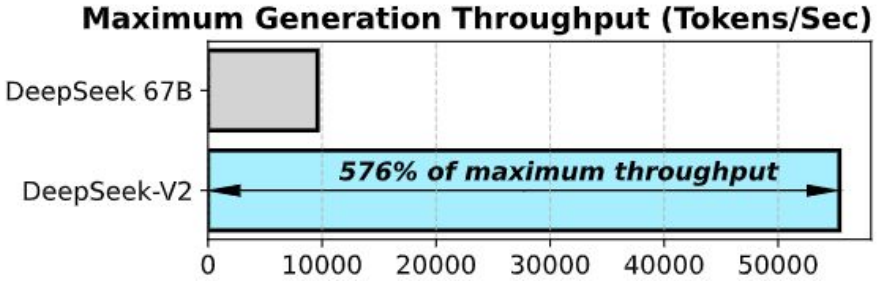
DeepSeek-V3 Technical Report

DeepSeek Performance

Saved 42.5% in training costs
Reduced KV cache by 93.35
Throughput to 576%







References:
[2405.04434] DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model

DeepSeek-V3 Technical Report

What's next in attention?

Thank you!

Key References and Links

Papers:

Covering,

Neural Machine Translation by Jointly Learning to Align and Translate https://arxiv.org/abs/1409.0473

Attention Is All You Need https://arxiv.org/abs/1706.03762

FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness https://arxiv.org/abs/2205.14135

Referencing,

Sequence to Sequence Learning with Neural Networks https://arxiv.org/abs/1409.3215

Youtube Links:

Attention is all you need (Transformer) - Model explanation (including math), Inference and Training https://www.youtube.com/watch?v=bCz4OMemCcA&t=2198s

Stanford CS25: V2 I Introduction to Transformers w/ Andrej Karpathy https://www.youtube.com/watch?v=XfpMkf4rD6E

Self Attention in Transformer Neural Networks https://www.youtube.com/watch?v=QCJQG4DuHT0&list=LL&index=7&t=662s