Advanced Methods in Natural Language Processing

Session 5: Attention, Transformers, and BERT

Arnault Gombert

May 2025

Barcelona School of Economics

Introduction

Already Old Fashion NLP

Course Overview:

- Explore the 2018-2022 cutting-edge techniques in Natural Language Processing.
- Gain hands-on experience with attention mechanisms, Transformers, and BERT.
- Understand the theory and practical application of advanced NLP models.

Learning Objectives:

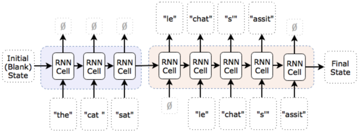
- Understand the limitations of traditional NLP models and how attention mechanisms address them.
- Dive deep into the architecture and functionalities of Transformer models.
- Master the BERT architecture, its pre-training and fine-tuning

Attention Process

Limitations of RNNs in Sequence-to-Sequence Models

Challenges in Long Sequences:

- Seq2Seq models encode the whole input sequence into **one** fixed-size vector.

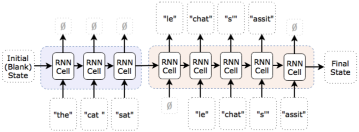


Seq2seq model architecture
Credit: R2Rt blog

Limitations of RNNs in Sequence-to-Sequence Models

Challenges in Long Sequences:

- Seq2Seq models encode the whole input sequence into **one** fixed-size vector.
- The decoder generates output from this single context vector.

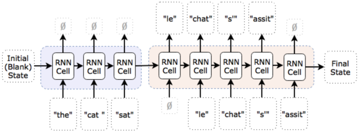


Seq2seq model architecture
Credit: R2Rt blog

Limitations of RNNs in Sequence-to-Sequence Models

Challenges in Long Sequences:

- Seq2Seq models encode the whole input sequence into **one** fixed-size vector.
- The decoder generates output from this single context vector.
- **Information Bottleneck**: The final state may not catch all nuances, especially long sequences.

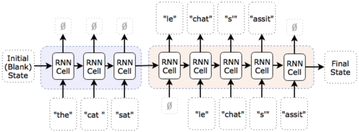


Seq2seq model architecture
Credit: R2Rt blog

Limitations of RNNs in Sequence-to-Sequence Models

Challenges in Long Sequences:

- Seq2Seq models encode the whole input sequence into **one** fixed-size vector.
- The decoder generates output from this single context vector.
- **Information Bottleneck**: The final state may not catch all nuances, especially long sequences.
- **Loss of Temporal Information**: Earlier inputs have less impact: potential context loss.

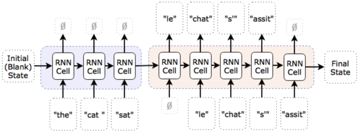


Seq2seq model architecture
Credit: R2Rt blog

Limitations of RNNs in Sequence-to-Sequence Models

Challenges in Long Sequences:

- Seq2Seq models encode the whole input sequence into **one** fixed-size vector.
- The decoder generates output from this single context vector.
- **Information Bottleneck**: The final state may not catch all nuances, especially long sequences.
- **Loss of Temporal Information**: Earlier inputs have less impact: potential context loss.

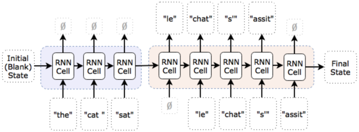


Seq2seq model architecture
Credit: R2Rt blog

Limitations of RNNs in Sequence-to-Sequence Models

Challenges in Long Sequences:

- Seq2Seq models encode the whole input sequence into **one** fixed-size vector.
- The decoder generates output from this single context vector.
- **Information Bottleneck**: The final state may not catch all nuances, especially long sequences.
- **Loss of Temporal Information**: Earlier inputs have less impact: potential context loss.



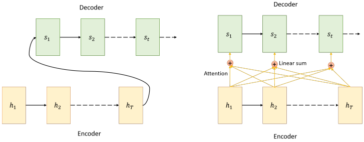
Seq2seq model architecture
Credit: R2Rt blog

Need for Enhanced Mechanism: We need a mechanism that addresses those issues by allowing the model to focus on different parts of the input sequence at each step of the output generation.

Beyond the Final State: Embracing Attention

Expanding the Contextual Horizon:

- **Full Sequence Utilization**: Instead of relying only the final state, attention mechanisms consider the entire sequence of hidden states.



(a) Vanilla Encoder Decoder Architecture

(b) Attention Mechanism

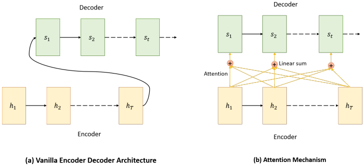
Visualization of attention

Credits: Shashank Yadav

Beyond the Final State: Embracing Attention

Expanding the Contextual Horizon:

- **Full Sequence Utilization**: Instead of relying only the final state, attention mechanisms consider the entire sequence of hidden states.
- **Dynamic Contextual Focus**: At each step, the model dynamically selects which parts of the sequence to emphasize.



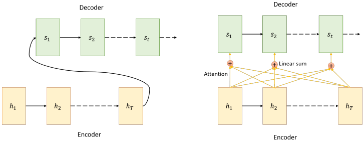
Visualization of attention

Credits: Shashank Yadav

Beyond the Final State: Embracing Attention

Expanding the Contextual Horizon:

- **Full Sequence Utilization**: Instead of relying only the final state, attention mechanisms consider the entire sequence of hidden states.
- **Dynamic Contextual Focus**: At each step, the model dynamically selects which parts of the sequence to emphasize.
- **Soft Memory Concept**: This approach is akin to having a 'soft memory' that retains and accesses all past states, enhancing the model's contextual understanding.



(a) Vanilla Encoder Decoder Architecture

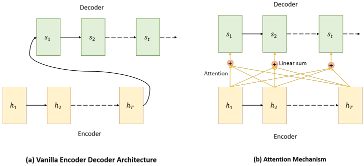
(b) Attention Mechanism

Visualization of attention
Credits: Shashank Yadav

Beyond the Final State: Embracing Attention

Expanding the Contextual Horizon:

- **Full Sequence Utilization**: Instead of relying only the final state, attention mechanisms consider the entire sequence of hidden states.
- **Dynamic Contextual Focus**: At each step, the model dynamically selects which parts of the sequence to emphasize.
- **Soft Memory Concept**: This approach is akin to having a 'soft memory' that retains and accesses all past states, enhancing the model's contextual understanding.

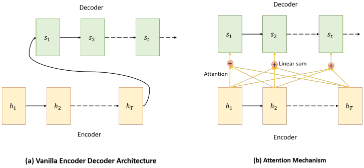


Visualization of attention
Credits: Shashank Yadav

Beyond the Final State: Embracing Attention

Expanding the Contextual Horizon:

- **Full Sequence Utilization**: Instead of relying only the final state, attention mechanisms consider the entire sequence of hidden states.
- **Dynamic Contextual Focus**: At each step, the model dynamically selects which parts of the sequence to emphasize.
- **Soft Memory Concept**: This approach is akin to having a 'soft memory' that retains and accesses all past states, enhancing the model's contextual understanding.



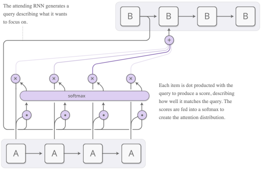
Visualization of attention
Credits: Shashank Yadav

Benefits of Attention: Attention provides a more nuanced and flexible way to represent sequences, enabling models to capture complex dependencies and relationships within the data.

Attention Mechanism - Theoretical Details

In translation tasks, you focus on relevant words and their context. The attention mechanism too by weighting inputs importance:

- **State Concatenation**: Hidden states **H**.

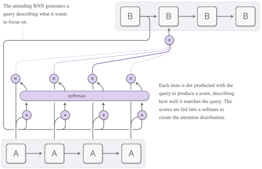


Credits: Olah & Carter, 2016

Attention Mechanism - Theoretical Details

In translation tasks, you focus on relevant words and their context. The attention mechanism too by weighting inputs importance:

- **State Concatenation**: Hidden states **H**.
- **Query Matrix**: Relevance of each part of the input **Q**.

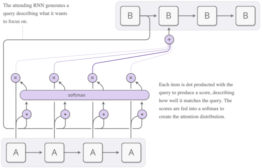


Credits: Olah & Carter, 2016

Attention Mechanism - Theoretical Details

In translation tasks, you focus on relevant words and their context. The attention mechanism too by weighting inputs importance:

- **State Concatenation**: Hidden states **H**.
- **Query Matrix**: Relevance of each part of the input **Q**.
- **Attention Scores**: Compute alignment scores between **Q** and **H** - product/FFNN.

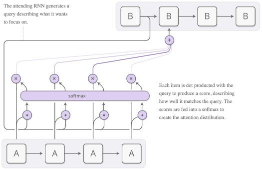


Credits: Olah & Carter, 2016

Attention Mechanism - Theoretical Details

In translation tasks, you focus on relevant words and their context. The attention mechanism too by weighting inputs importance:

- **State Concatenation**: Hidden states **H**.
- **Query Matrix**: Relevance of each part of the input **Q**.
- **Attention Scores**: Compute alignment scores between **Q** and **H** - product/FFNN.
- **Softmax Normalization**: Softmax on scores to get attention weights.

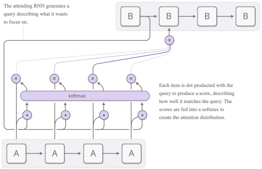


Credits: Olah & Carter, 2016

Attention Mechanism - Theoretical Details

In translation tasks, you focus on relevant words and their context. The attention mechanism too by weighting inputs importance:

- **State Concatenation**: Hidden states **H**.
- **Query Matrix**: Relevance of each part of the input **Q**.
- **Attention Scores**: Compute alignment scores between **Q** and **H** - product/FFNN.
- **Softmax Normalization**: Softmax on scores to get attention weights.
- **Context Vector**: Weighted sum of the hidden states **H**.

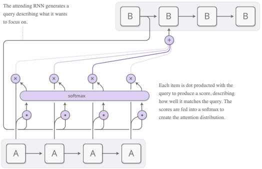


Credits: Olah & Carter, 2016

Attention Mechanism - Theoretical Details

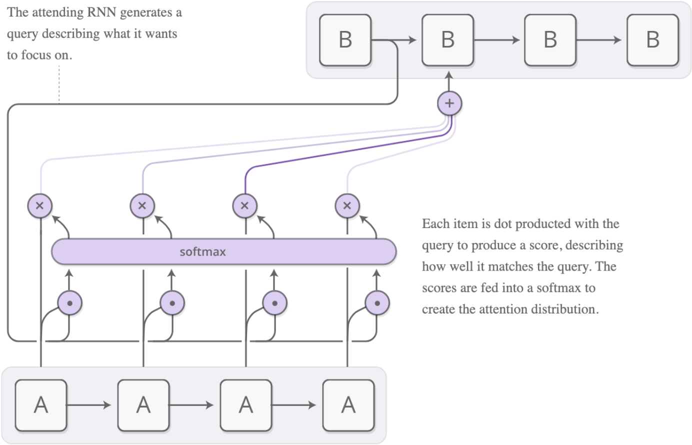
In translation tasks, you focus on relevant words and their context. The attention mechanism too by weighting inputs importance:

- **State Concatenation**: Hidden states **H**.
- **Query Matrix**: Relevance of each part of the input **Q**.
- **Attention Scores**: Compute alignment scores between **Q** and **H** - product/FFNN.
- **Softmax Normalization**: Softmax on scores to get attention weights.
- **Context Vector**: Weighted sum of the hidden states **H**.
- **Decoder Input**: Feed the context vector, ie. the attentive readout of the input.



Credits: Olah & Carter, 2016

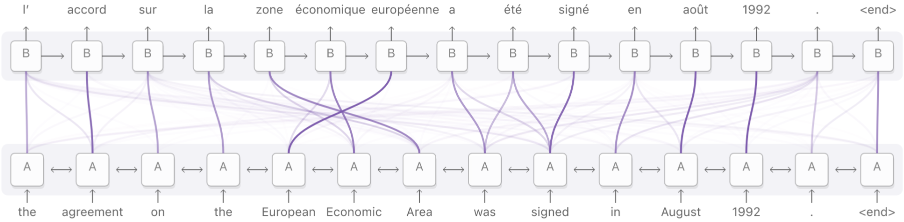
Attention Mechanism - Theoretical Details



Credits: Olah & Carter, 2016

This process allows the model to dynamically focus on different parts of the input sequence, improving its ability to capture relevant context.

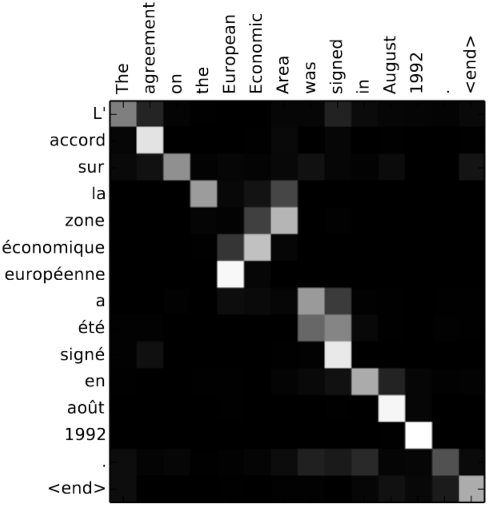
Illustrated Attention



Attention illustration

Olah & Carter, 2016

Illustrated Attention



Attention Matrix

Bahdanau et al. (2014)

Limitations of RNNs (with attention too!)

Beyond RNNs and Attention Mechanisms:

- While RNNs (with or without attention) have been pivotal in handling sequences, they inherently process data sequentially, leading to limitations in parallelization and computational efficiency.

Limitations of RNNs (with attention too!)

Beyond RNNs and Attention Mechanisms:

- While RNNs (with or without attention) have been pivotal in handling sequences, they inherently process data sequentially, leading to limitations in parallelization and computational efficiency.
- Attention mechanisms significantly improve the ability of models to focus on relevant parts of the input. However, the sequential nature of RNNs still poses challenges in capturing long-distance dependencies.

RNN Limitations: Lack of Parallelization

Sequential Nature of RNNs:

- RNNs process sequences one element at a time.
- This sequential dependency forms a chain-like structure.

Example: Sentence Processing

- Consider processing the sentence: "The cat sat on the mat."
- RNNs process each word sequentially, process "The" to process "cat," then "sat," and so on.
- This characteristic makes it difficult to leverage modern hardware's parallel processing capabilities (GPU!): longer training and inference times.

Implication: The inability to process elements in parallel significantly hampers the efficiency of RNNs, especially for long sequences where the computational graph becomes excessively extended.

Long-Distance Dependencies in Augmented RNNs

Challenge of Capturing Long-Distance Dependencies:

- RNNs struggle to capture dependencies between elements that are far apart in the sequence.
- Remember gradients vanishing or exploding?

Limitation with Attention:

- Attention mechanisms, while providing focus on relevant inputs, still has a sequential nature and so.. associated gradient issues.
- Generally have a finite contextual window, limiting the capture and utilization of information from distant elements.
- The sequential computation still influences the representation of each element: affects capacity in handling long-range dependencies.

Example: Contextual Ambiguity in Text

Consider a complex sentence with crucial context at the end.

Transformers: A Paradigm Shift

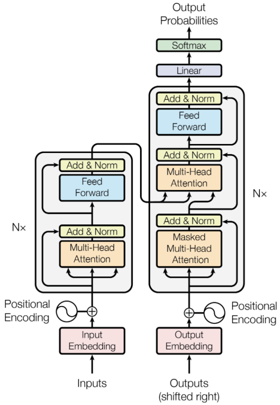
Transformers: Revolutionizing Sequence Processing

Transformers, introduced in "Attention is All You Need" by Vaswani et al. (2017) (100k+ citations !), marked a turning point in sequence processing:

Transformers: Revolutionizing Sequence Processing

Transformers, introduced in "Attention is All You Need" by Vaswani et al. (2017) (100k+ citations !), marked a turning point in sequence processing:

- **Parallelization**- Eliminate sequential computation: much faster training/inferences.



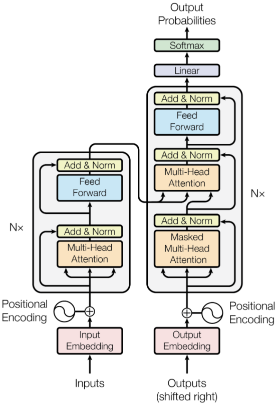
Transformer Architecture

Credit: Vaswani et al. (2017)

Transformers: Revolutionizing Sequence Processing

Transformers, introduced in "Attention is All You Need" by Vaswani et al. (2017) (100k+ citations !), marked a turning point in sequence processing:

- **Parallelization**- Eliminate sequential computation: much faster training/inferences.
- **Positional Encoding**- Add word order info: better context understanding.

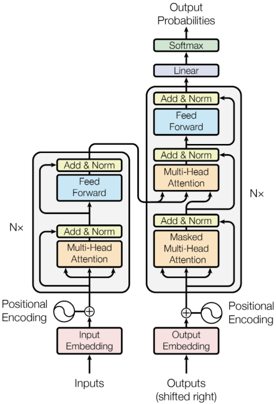


Transformer Architecture
Credit: Vaswani et al. (2017)

Transformers: Revolutionizing Sequence Processing

Transformers, introduced in "Attention is All You Need" by Vaswani et al. (2017) (100k+ citations !), marked a turning point in sequence processing:

- **Parallelization**- Eliminate sequential computation: much faster training/inferences.
- **Positional Encoding**- Add word order info: better context understanding.
- **Self-Attention**- Enable the model to dynamically weigh each word importance.



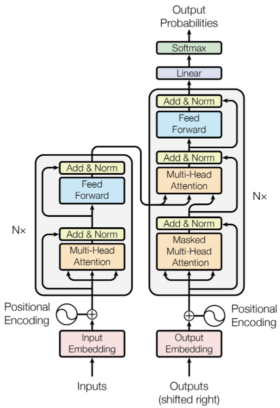
Transformer Architecture

Credit: Vaswani et al. (2017)

Transformers: Revolutionizing Sequence Processing

Transformers, introduced in "Attention is All You Need" by Vaswani et al. (2017) (100k+ citations !), marked a turning point in sequence processing:

- **Parallelization**- Eliminate sequential computation: much faster training/inferences.
- **Positional Encoding**- Add word order info: better context understanding.
- **Self-Attention**- Enable the model to dynamically weigh each word importance.
- **Benchmark Performance**- Set new SOA results, particularly in machine translation tasks.

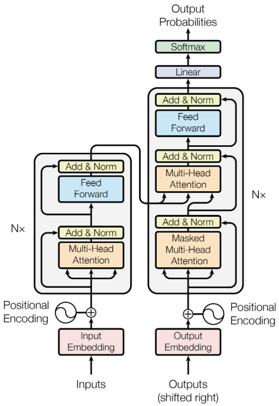


Transformer Architecture
Credit: Vaswani et al. (2017)

Transformers: Revolutionizing Sequence Processing

Transformers, introduced in "Attention is All You Need" by Vaswani et al. (2017) (100k+ citations !), marked a turning point in sequence processing:

- **Parallelization**- Eliminate sequential computation: much faster training/inferences.
- **Positional Encoding**- Add word order info: better context understanding.
- **Self-Attention**- Enable the model to dynamically weigh each word importance.
- **Benchmark Performance**- Set new SOA results, particularly in machine translation tasks.

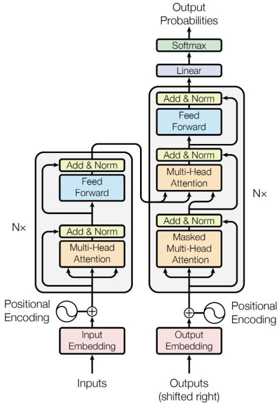


Transformer Architecture
Credit: Vaswani et al. (2017)

Transformers: Revolutionizing Sequence Processing

Transformers, introduced in "Attention is All You Need" by Vaswani et al. (2017) (100k+ citations !), marked a turning point in sequence processing:

- **Parallelization**- Eliminate sequential computation: much faster training/inferences.
- **Positional Encoding**- Add word order info: better context understanding.
- **Self-Attention**- Enable the model to dynamically weigh each word importance.
- **Benchmark Performance**- Set new SOA results, particularly in machine translation tasks.



Transformer Architecture
Credit: Vaswani et al. (2017)

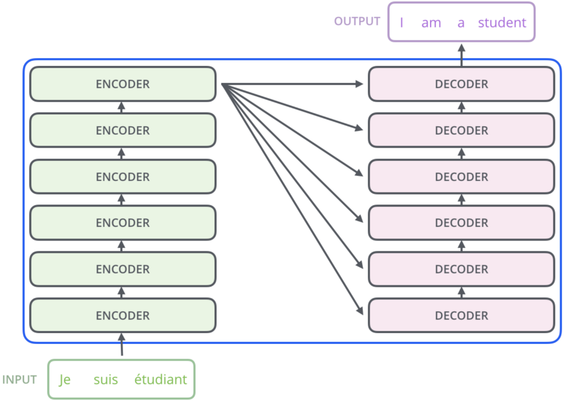
This breakthrough laid the foundation for subsequent advances like BERT, GPT-3, and other large language models, continuously pushing the boundaries of NLP.

Transformer: A High-Level Overview

The Essence of Transformers:
At its core, the Transformer model is designed for tasks like machine translation, taking a sentence in one language and outputting its translation in another.

- It consists of two main components: an **encoding component** and a **decoding component**
- The **encoding component**, composed of several layers processes the input sentence, capturing its meaning and context into an internal representation.
- The **decoding component**, composed of several layers, then generates the translated output, one word at a time, based on the encoded representation and what it has generated so far.
- Connections between the encoder and decoder allow the model to focus on relevant parts of the input sentence during each step of the output generation.

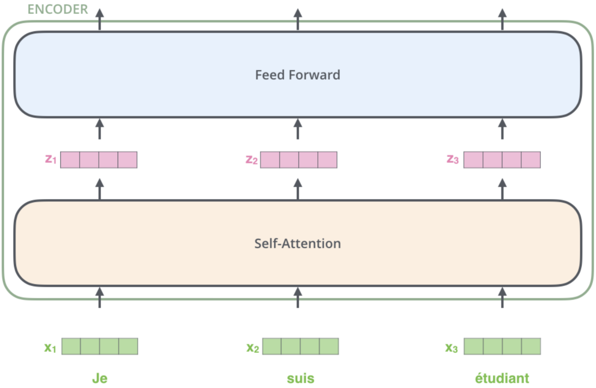
Transformer: A High-Level Overview



Encoders and Decoders stacked

Credit: Alammar (2020)

Zoom on encoder



Encoder's decomposition

Credit: Alammar (2020)

Key Property: Path Independence and Parallelization

Distinct Paths in the Encoder:

- Each word flows through its own path, maintaining a unique state that captures its context within the sequence.

Parallelization in Feed-Forward Layers:

Key Property: Path Independence and Parallelization

Distinct Paths in the Encoder:

- Each word flows through its own path, maintaining a unique state that captures its context within the sequence.
- Self-attention introduces dependencies: each word relates to the others.

Parallelization in Feed-Forward Layers:

Key Property: Path Independence and Parallelization

Distinct Paths in the Encoder:

- Each word flows through its own path, maintaining a unique state that captures its context within the sequence.
- Self-attention introduces dependencies: each word relates to the others.

Parallelization in Feed-Forward Layers:

- Unlike the self-attention layer, FFN processes each position independently.

Key Property: Path Independence and Parallelization

Distinct Paths in the Encoder:

- Each word flows through its own path, maintaining a unique state that captures its context within the sequence.
- Self-attention introduces dependencies: each word relates to the others.

Parallelization in Feed-Forward Layers:

- Unlike the self-attention layer, FFN processes each position independently.
- This independence enables parallel computation of the FFN: enhancies the model's efficiency and training speed.

Key Property: Path Independence and Parallelization

Distinct Paths in the Encoder:

- Each word flows through its own path, maintaining a unique state that captures its context within the sequence.
- Self-attention introduces dependencies: each word relates to the others.

Parallelization in Feed-Forward Layers:

- Unlike the self-attention layer, FFN processes each position independently.
- This independence enables parallel computation of the FFN: enhancies the model's efficiency and training speed.
- The Transformer leverages this architecture to parallelize operations, a stark contrast to the sequential nature of traditional RNNs.

Key Property: Path Independence and Parallelization

Distinct Paths in the Encoder:

- Each word flows through its own path, maintaining a unique state that captures its context within the sequence.
- Self-attention introduces dependencies: each word relates to the others.

Parallelization in Feed-Forward Layers:

- Unlike the self-attention layer, FFN processes each position independently.
- This independence enables parallel computation of the FFN: enhancies the model's efficiency and training speed.
- The Transformer leverages this architecture to parallelize operations, a stark contrast to the sequential nature of traditional RNNs.

Key Property: Path Independence and Parallelization

Distinct Paths in the Encoder:

- Each word flows through its own path, maintaining a unique state that captures its context within the sequence.
- Self-attention introduces dependencies: each word relates to the others.

Parallelization in Feed-Forward Layers:

- Unlike the self-attention layer, FFN processes each position independently.
- This independence enables parallel computation of the FFN: enhancies the model's efficiency and training speed.
- The Transformer leverages this architecture to parallelize operations, a stark contrast to the sequential nature of traditional RNNs.

Conclusion: Processing each word independently in FF layers and inter-word relationships modeled in self-attention, strikes a balance between contextual understanding and computational efficiency.

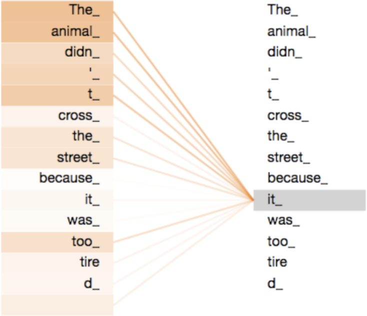
Understanding Self-Attention Mechanism

Core Concept of Self-Attention:

- Self-attention, a crucial component of the Transformer, allows each token in the input sequence to interact with every other token, capturing complex word relationships and dependencies.
- This mechanism enables the model to dynamically focus on different parts of the input, enhancing its ability to understand and generate contextually rich text.
- For a deeper dive into the Transformer's architecture, refer to "The Illustrated Transformer" by Alammar (2020).

Understanding Self-Attention Mechanism

Visualizing Self-Attention: Below is an illustration of how self-attention operates on an example sentence. Notice how the encoding of each token involves consideration of the entire sequence, allowing the model to integrate context effectively.



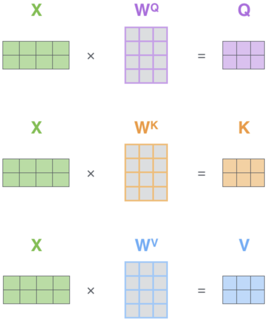
Self-Attention: Focus on the word *it*

Credit: Alammar (2020)

Understanding Self-Attention

Self-Attention Vectors:

- For each input, self-attention generates three vectors: **Query, Q**, **Key, K**, and **Value, V**.



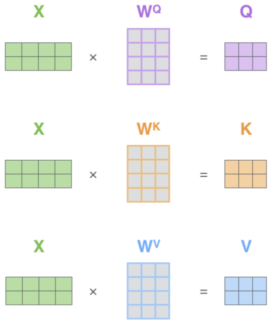
Q, K, V in Self-Attention

Credit: Alammar (2020)

Understanding Self-Attention

Self-Attention Vectors:

- For each input, self-attention generates three vectors: **Query, Q**, **Key, K**, and **Value, V**.
- These vectors comes from multiplying the input by three matrices (trained too).



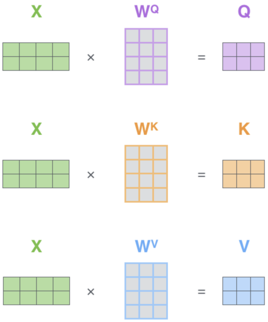
Q, K, V in Self-Attention

Credit: Alammar (2020)

Understanding Self-Attention

Self-Attention Vectors:

- For each input, self-attention generates three vectors: **Query, Q**, **Key, K**, and **Value, V**.
- These vectors comes from multiplying the input by three matrices (trained too).
- **Q** represents the attention focus.



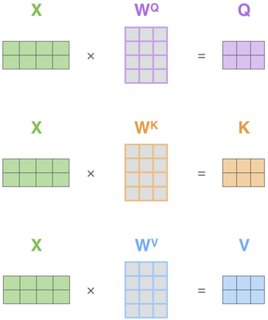
Q, K, V in Self-Attention

Credit: Alammar (2020)

Understanding Self-Attention

Self-Attention Vectors:

- For each input, self-attention generates three vectors: **Query, Q**, **Key, K**, and **Value, V**.
- These vectors comes from multiplying the input by three matrices (trained too).
- **Q** represents the attention focus.
- **K** and **V** are representations of the input, providing the context for each word.

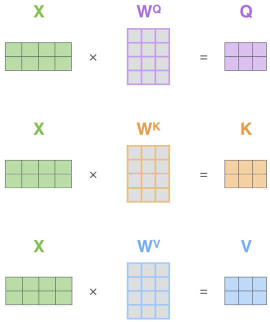


Q, K, V in Self-Attention
Credit: Alammar (2020)

Understanding Self-Attention

Self-Attention Vectors:

- For each input, self-attention generates three vectors: **Query, Q**, **Key, K**, and **Value, V**.
- These vectors comes from multiplying the input by three matrices (trained too).
- **Q** represents the attention focus.
- **K** and **V** are representations of the input, providing the context for each word.
- In the *encoder*: Q, K, and V are projections of the input embeddings.

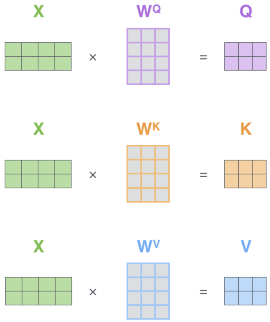


Q, K, V in Self-Attention
Credit: Alammar (2020)

Understanding Self-Attention

Self-Attention Vectors:

- For each input, self-attention generates three vectors: **Query, Q**, **Key, K**, and **Value, V**.
- These vectors comes from multiplying the input by three matrices (trained too).
- **Q** represents the attention focus.
- **K** and **V** are representations of the input, providing the context for each word.
- In the *encoder*: Q, K, and V are projections of the input embeddings.
- In the *decoder*: K and V come from the encoder's output, while Q comes from the previous decoder layer.

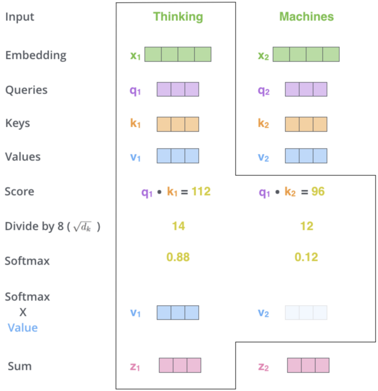


Q, K, V in Self-Attention
Credit: Alammar (2020)

Self-Attention: Calculating Attention Scores

Calculating Attention Scores:

- For each token i, calculate scores with the dot product of q_i with all key vectors in K.



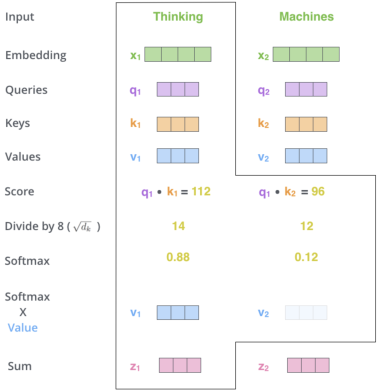
Attention score calculation

Credit: Alammar (2020)

Self-Attention: Calculating Attention Scores

Calculating Attention Scores:

- For each token i, calculate scores with the dot product of q_i with all key vectors in K.
- Normalize the scores by dimension of key vectors to stabilize gradients.



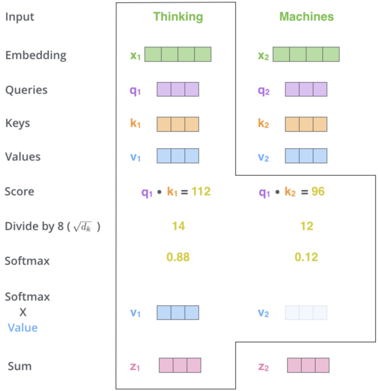
Attention score calculation

Credit: Alammar (2020)

Self-Attention: Calculating Attention Scores

Calculating Attention Scores:

- For each token i, calculate scores with the dot product of q_i with all key vectors in K.
- Normalize the scores by dimension of key vectors to stabilize gradients.
- Apply softmax to normalized scores, yielding a distribution that quantifies the relevance of each token's contribution to the representation of token i.



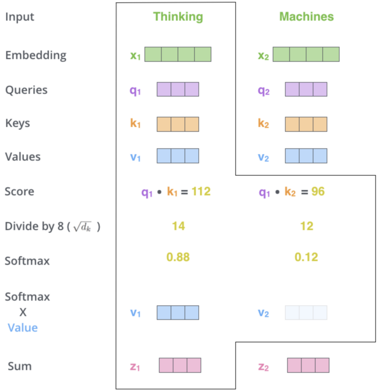
Attention score calculation

Credit: Alammar (2020)

Self-Attention: Calculating Attention Scores

Calculating Attention Scores:

- For each token i, calculate scores with the dot product of q_i with all key vectors in K.
- Normalize the scores by dimension of key vectors to stabilize gradients.
- Apply softmax to normalized scores, yielding a distribution that quantifies the relevance of each token's contribution to the representation of token i.



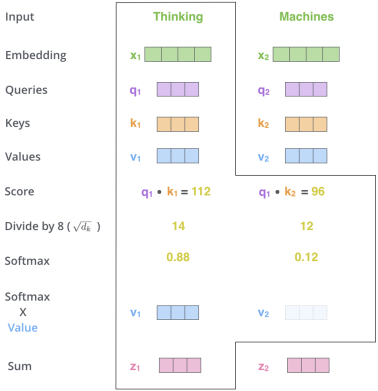
Attention score calculation

Credit: Alammar (2020)

Self-Attention: Calculating Attention Scores

Calculating Attention Scores:

- For each token i, calculate scores with the dot product of q_i with all key vectors in K.
- Normalize the scores by dimension of key vectors to stabilize gradients.
- Apply softmax to normalized scores, yielding a distribution that quantifies the relevance of each token's contribution to the representation of token i.



Attention score calculation

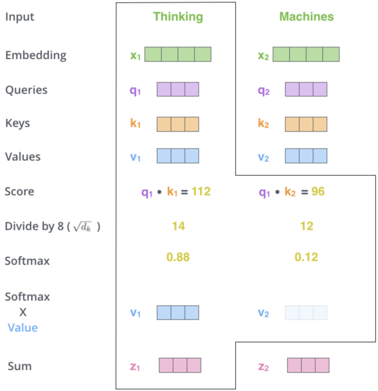
Credit: Alammar (2020)

Intuition: This process allows the model to dynamically allocate focus, placing more weight on relevant tokens, as determined by the context within the sequence.

Self-Attention: Obtaining Attended Representation

Forming Attended Representation:

- Multiply each value vector by its corresponding score, emphasizing vectors with higher relevance.



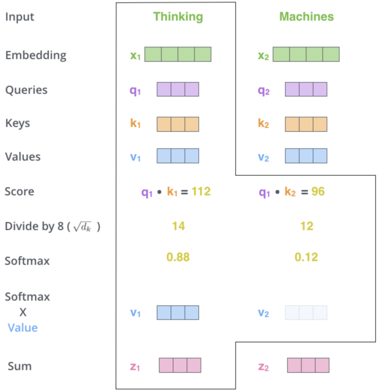
Weighted sum of value vectors

Credit: Alammar (2020)

Self-Attention: Obtaining Attended Representation

Forming Attended Representation:

- Multiply each value vector by its corresponding score, emphasizing vectors with higher relevance.
- Sum weighted values to get the final attended representation for token *i*. It encapsulates the contextual information.

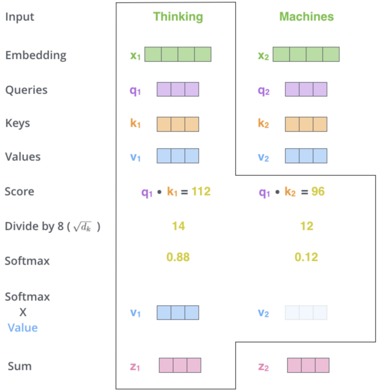


Weighted sum of value vectors

Credit: Alammar (2020)

Forming Attended Representation:

- Multiply each value vector by its corresponding score, emphasizing vectors with higher relevance.
- Sum weighted values to get the final attended representation for token *i*. It encapsulates the contextual information.
- **Result:** The output vector *i* is a synthesized representation integrating contextual information from the entire sequence, ready to be fed into subsequent layers.

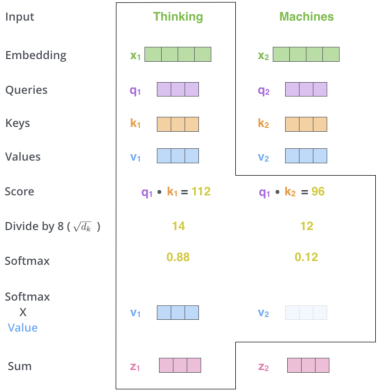


Weighted sum of value vectors
Credit: Alammar (2020)

Self-Attention: Obtaining Attended Representation

Forming Attended Representation:

- Multiply each value vector by its corresponding score, emphasizing vectors with higher relevance.
- Sum weighted values to get the final attended representation for token *i*. It encapsulates the contextual information.
- **Result:** The output vector *i* is a synthesized representation integrating contextual information from the entire sequence, ready to be fed into subsequent layers.

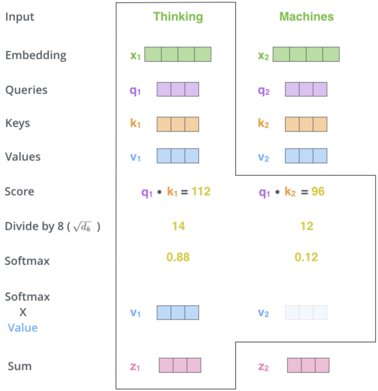


Weighted sum of value vectors
Credit: Alammar (2020)

Self-Attention: Obtaining Attended Representation

Forming Attended Representation:

- Multiply each value vector by its corresponding score, emphasizing vectors with higher relevance.
- Sum weighted values to get the final attended representation for token *i*. It encapsulates the contextual information.
- **Result:** The output vector *i* is a synthesized representation integrating contextual information from the entire sequence, ready to be fed into subsequent layers.



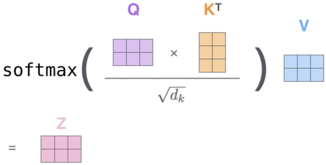
Weighted sum of value vectors
Credit: Alammar (2020)

Efficiency in Implementation: The process is not sequential and the implementation leverages matrix operations for efficient computation to handle entire sequences simultaneously.

Matrix Representation in Self-Attention

Matrix Formulation in Transformers:

- Input embeddings matrix X is multiplied by trained weight matrices (Wq, Wk, Wv) to obtain Q, K, V matrices.



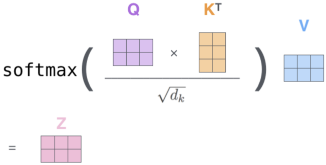
Self-Attention Mechanism

Credit: Alammar (2020)

Matrix Representation in Self-Attention

Matrix Formulation in Transformers:

- Input embeddings matrix X is multiplied by trained weight matrices (Wq, Wk, Wv) to obtain Q, K, V matrices.
- Transformer architecture 6 layers for both encoding and decoding: the model can capture complex relationships.



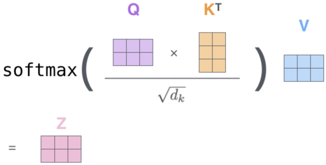
Self-Attention Mechanism

Credit: Alammar (2020)

Matrix Representation in Self-Attention

Matrix Formulation in Transformers:

- Input embeddings matrix X is multiplied by trained weight matrices (Wq, Wk, Wv) to obtain Q, K, V matrices.
- Transformer architecture 6 layers for both encoding and decoding: the model can capture complex relationships.
- **Multi-Head Attention:**



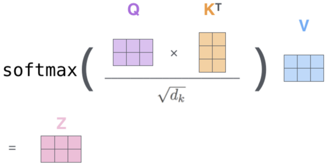
Self-Attention Mechanism

Credit: Alammar (2020)

Matrix Representation in Self-Attention

Matrix Formulation in Transformers:

- Input embeddings matrix X is multiplied by trained weight matrices (Wq, Wk, Wv) to obtain Q, K, V matrices.
- Transformer architecture 6 layers for both encoding and decoding: the model can capture complex relationships.
- **Multi-Head Attention:**
  - Explores different representation sub-spaces.



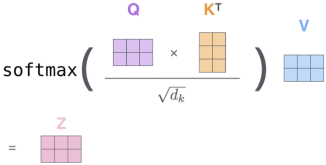
Self-Attention Mechanism

Credit: Alammar (2020)

Matrix Representation in Self-Attention

Matrix Formulation in Transformers:

- Input embeddings matrix X is multiplied by trained weight matrices (Wq, Wk, Wv) to obtain Q, K, V matrices.
- Transformer architecture 6 layers for both encoding and decoding: the model can capture complex relationships.
- **Multi-Head Attention:**
  - Explores different representation sub-spaces.
  - Outputs of different heads are concatenated.



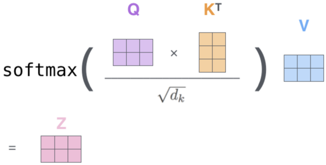
Self-Attention Mechanism

Credit: Alammar (2020)

Matrix Representation in Self-Attention

Matrix Formulation in Transformers:

- Input embeddings matrix X is multiplied by trained weight matrices (Wq, Wk, Wv) to obtain Q, K, V matrices.
- Transformer architecture 6 layers for both encoding and decoding: the model can capture complex relationships.
- **Multi-Head Attention:**
  - Explores different representation sub-spaces.
  - Outputs of different heads are concatenated.



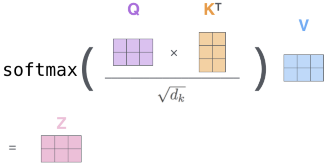
Self-Attention Mechanism

Credit: Alammar (2020)

Matrix Representation in Self-Attention

Matrix Formulation in Transformers:

- Input embeddings matrix X is multiplied by trained weight matrices (Wq, Wk, Wv) to obtain Q, K, V matrices.
- Transformer architecture 6 layers for both encoding and decoding: the model can capture complex relationships.
- **Multi-Head Attention:**
  - Explores different representation sub-spaces.
  - Outputs of different heads are concatenated.



Self-Attention Mechanism
Credit: Alammar (2020)

Efficiency: The use of matrix operations condenses the calculation: the model processes inputs in parallel, increasing efficiency and speed.

Transformers: Impact and Achievements

Revolutionizing Performance:

- Transformers have consistently **outperformed SOA models** in machine translation, showcasing their ability to understand and generate language effectively.
- Their architecture allows for **adaptation across a variety of NLP tasks**, such as English parsing, sentiment analysis, and more.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Impact of Transformers
Credit: Vaswani et al. (2019)

Transformers: Impact and Achievements

Parallelization and Efficiency:

- The non-sequential nature eliminates the need for sequential data processing, allowing for **parallel computation**.
- This architectural innovation makes Transformers well-suited for training on **GPUs and TPUs**, reducing training time.

Pioneering New Research Directions:

- By overcoming previous limitations, Transformers have **opened new avenues for research** and application in NLP, leading to the development of models like BERT, GPT-3, and others.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$\mathbf{3.3 \cdot 10^{18}}$	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

Impact of Transformers

Credit: Vaswani et al. (2019)

BERT Model

Introduction to BERT

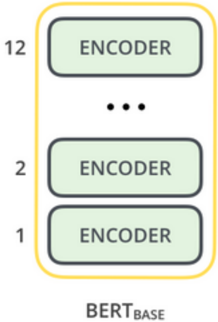
BERT - A Milestone in NLP:

- **Background**: Bidirectional Encoder Representations from Transformers is a groundbreaking model introduced by Devlin et al. in 2019 (90k+ citations!).
- **Bidirectional Context**: BERT captures context from both directions (left and right) for every token in a sequence, offering a deeper understanding of language structure.
- **Pre-training on Language Understanding**: BERT is pre-trained on a large corpus, enabling it to develop a rich understanding of language patterns and structures.
- **Fine-tuning for Specific Tasks**: After pre-training, BERT can be fine-tuned with just one additional output layer to create SOA models for a wide range of tasks, such as QA, sentiment analysis, and more.

BERT Architecture Overview

BERT's architecture is a rooster on hormones:

- **Stacked Encoder Layers**: BERT stacks multiple layers of transformer encoders.
- **Two Model Variants** introduced:
  - **BERT-Base**: 12 encoders.
  - **BERT-Large**: 24 encoders.
- **Dimensionality**: The hidden size is increased to 768 dimensions, compared to the 512 in the original Transformer model.
- **Attention Heads**: Features 12 self-attention heads, to get more nuanced context.
- **Training Scale**: trained during 4 days on 4 TPUs !



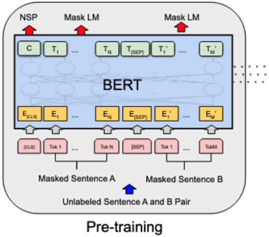
BERT Model Architecture
Credit: Alammar (2019)

Training Procedure

BERT Pre-training: Language Modeling

Masked Language Model (MLM):

- Utilizes bi-directional context by **randomly masking 15% of the tokens** in each sequence.

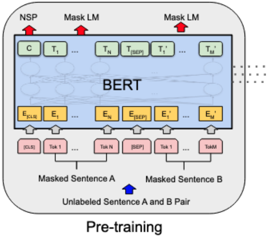


Masked Language Model
Credit: Devlin et al. (2019)

BERT Pre-training: Language Modeling

Masked Language Model (MLM):

- Utilizes bi-directional context by **randomly masking 15% of the tokens** in each sequence.
- Of the masked tokens:

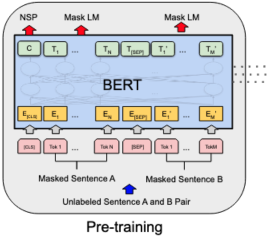


Masked Language Model
Credit: Devlin et al. (2019)

BERT Pre-training: Language Modeling

Masked Language Model (MLM):

- Utilizes bi-directional context by **randomly masking 15% of the tokens** in each sequence.
- Of the masked tokens:
  - 80% are replaced with [MASK].

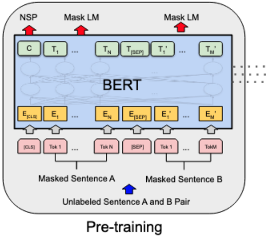


Masked Language Model
Credit: Devlin et al. (2019)

BERT Pre-training: Language Modeling

Masked Language Model (MLM):

- Utilizes bi-directional context by **randomly masking 15% of the tokens** in each sequence.
- Of the masked tokens:
    - 80% are replaced with [MASK].
    - 10% are replaced with a random token.

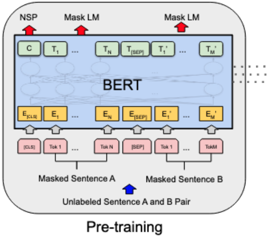


Masked Language Model
Credit: Devlin et al. (2019)

BERT Pre-training: Language Modeling

Masked Language Model (MLM):

- Utilizes bi-directional context by **randomly masking 15% of the tokens** in each sequence.
- Of the masked tokens:
  - 80% are replaced with [MASK].
  - 10% are replaced with a random token.
  - 10% remain unchanged.

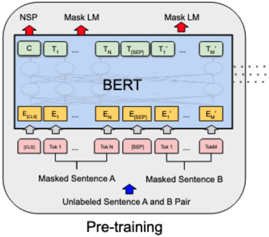


Masked Language Model
Credit: Devlin et al. (2019)

BERT Pre-training: Language Modeling

Masked Language Model (MLM):

- Utilizes bi-directional context by **randomly masking 15% of the tokens** in each sequence.
- Of the masked tokens:
  - 80% are replaced with [MASK].
  - 10% are replaced with a random token.
  - 10% remain unchanged.
- **Goal**: Predict the masked words based on context, ensuring the model doesn't rely on [MASK] during fine-tuning.



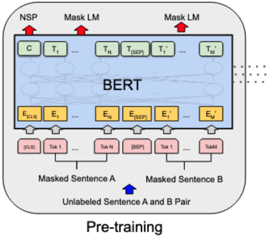
Masked Language Model

Credit: Devlin et al. (2019)

BERT Pre-training: Language Modeling

Masked Language Model (MLM):

- Utilizes bi-directional context by **randomly masking 15% of the tokens** in each sequence.
- Of the masked tokens:
    - 80% are replaced with [MASK].
    - 10% are replaced with a random token.
    - 10% remain unchanged.
- **Goal**: Predict the masked words based on context, ensuring the model doesn't rely on [MASK] during fine-tuning.
- **Loss Function**: Cross-entropy, measuring the model's performance in predicting the masked tokens.



Masked Language Model
Credit: Devlin et al. (2019)

Example of BERT Pre-training: Masked Language Modeling

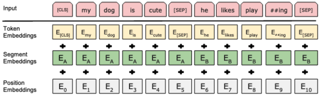
Masked Language Model (MLM) in Action: Imagine a sentence: ”The quick brown fox jumps over the lazy dog.”

- Randomly masking 15% of the tokens, e.g., ”The quick brown [MASK] jumps over the [MASK] dog.”
- Applying the masking rules:
  - ”The quick brown [MASK] jumps over the [MASK] dog.” (80% replaced with [MASK])
  - ”The quick brown cat jumps over the [MASK] dog.” (10% replaced with random token ”cat”)
  - ”The quick brown [MASK] jumps over the lazy dog.” (10% unchanged)
- BERT’s task: Predict ”fox” and ”lazy” from the context.

BERT Pre-training: Next Sentence Prediction

Understanding Sentence Relationships:

- Aims to teach BERT about the relationship between two sentences.



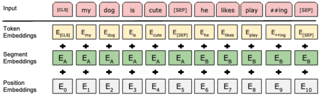
Next Sentence Prediction

Credit: Devlin et al. (2019)

BERT Pre-training: Next Sentence Prediction

Understanding Sentence Relationships:

- Aims to teach BERT about the relationship between two sentences.
- A binary classification task: Is the second sentence the actual next sentence in the original document?



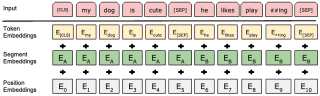
Next Sentence Prediction

Credit: Devlin et al. (2019)

BERT Pre-training: Next Sentence Prediction

Understanding Sentence Relationships:

- Aims to teach BERT about the relationship between two sentences.
- A binary classification task: Is the second sentence the actual next sentence in the original document?
- Training data:



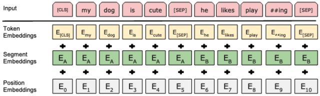
Next Sentence Prediction

Credit: Devlin et al. (2019)

BERT Pre-training: Next Sentence Prediction

Understanding Sentence Relationships:

- Aims to teach BERT about the relationship between two sentences.
- A binary classification task: Is the second sentence the actual next sentence in the original document?
- Training data:
  - 50% of the time, B presents the actual next sentence.



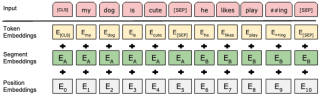
Next Sentence Prediction

Credit: Devlin et al. (2019)

BERT Pre-training: Next Sentence Prediction

Understanding Sentence Relationships:

- Aims to teach BERT about the relationship between two sentences.
- A binary classification task: Is the second sentence the actual next sentence in the original document?
- Training data:
  - 50% of the time, B presents the actual next sentence.
  - 50% of the time, a random sentence from the corpus is chosen as B.



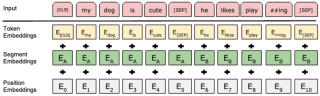
Next Sentence Prediction

Credit: Devlin et al. (2019)

BERT Pre-training: Next Sentence Prediction

Understanding Sentence Relationships:

- Aims to teach BERT about the relationship between two sentences.
- A binary classification task: Is the second sentence the actual next sentence in the original document?
- Training data:
  - 50% of the time, B presents the actual next sentence.
  - 50% of the time, a random sentence from the corpus is chosen as B.
- Uses special tokens ([CLS], [SEP], [END]) and sentence embeddings to differentiate sentences and perform classification.



Next Sentence Prediction

Credit: Devlin et al. (2019)

Example of BERT Pre-training: Next Sentence Prediction

Next Sentence Prediction (NSP) in Practice: Consider the sentence for BERT to analyze:
"The quick brown fox jumps over the lazy dog."

- Training instance creation:
  - Actual next sentence case: "They live happily ever after." (True next sentence)
  - Random sentence case: "Pizza is a popular dish in Italy." (Randomly chosen)
- BERT's task: Determine if the second sentence logically follows the first.

Special Tokens and Embeddings:

- Uses [CLS] at the beginning to signify the start of inputs.
- Token [SEP] separates the two sentences.
- Uses [END] at the beginning to signify the end of inputs.

BERT Tokenization Process

Tokenization and Special Tokens: BERT's tokenization process is crucial for understanding how it processes input data. Here's a breakdown:

Example: Given the input "The quick brown fox jumps over the lazy dog. What does the fox do?", the tokenization process would look something like:

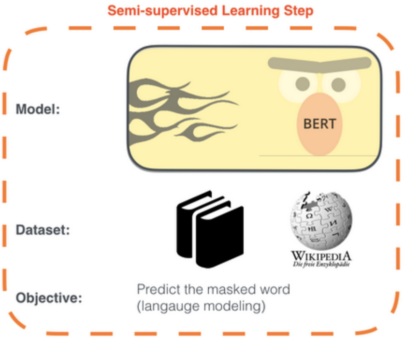
`[CLS] The quick brown fox jumps over the lazy dog [SEP] What does the fox do [SEP]`

BERT Pre-training on Large Corpora

Leveraging Massive Text Data: BERT's pre-training phase involves training on a large and diverse text corpus.

Benefits:

- BERT learns rich representations of language, capturing nuances, grammar, and relationships between words and sentences.
- The extensive pre-training enables BERT to be effectively fine-tuned for a wide range of specific tasks with relatively little task-specific data.



BERT pre-training process
Credit: Alammar (2019)

Fine-tuning BERT for Downstream Tasks

Classification Tasks:

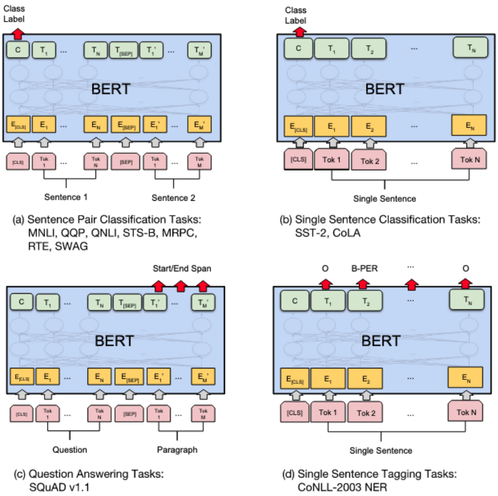
- The hidden state corresponding to the [CLS] token is used as the aggregate sequence representation for classification tasks.
- Additional layers can be added on top of BERT to fine-tune for specific classification objectives.

Token-Level Tasks (NER, QA..):

- BERT generates a representation for each token in the input.
- These representations are used for token-level predictions, enabling fine-grained tasks like named entity recognition or question answering.

Flexibility and Adaptability: BERT's design allows for straightforward adaptation to a wide range of NLP tasks, making it a versatile tool for many applications.

Fine-tuning BERT for Downstream Tasks



Fine-tuning BERT for specific tasks

Credit: Devlin et al. (2019)

BERT's Performance on GLUE Benchmark

Benchmarking BERT's Language Understanding:

- The GLUE benchmark is a collection of diverse natural language understanding tasks.
- BERT set new state-of-the-art records, showcasing its exceptional understanding of language nuances and contexts.
- The tasks include question answering, sentiment analysis, text similarity, and other complex language understanding challenges.

System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	**Average**
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

Table 1: GLUE Test results, scored by the evaluation server (https://gluebenchmark.com/leaderboard). The number below each task denotes the number of training examples. The "Average" column is slightly different than the official GLUE score, since we exclude the problematic WNLI set.[8] BERT and OpenAI GPT are single-model, single task. F1 scores are reported for QQP and MRPC, Spearman correlations are reported for STS-B, and accuracy scores are reported for the other tasks. We exclude entries that use BERT as one of their components.

BERT's Results on GLUE Benchmark - redit: Devlin et al. (2019)

Enhancements and Extensions of BERT

Following the introduction of BERT, subsequent research has proposed enhancements and variations, to refine and build upon its architecture:

- **RoBERTa (Liu et al., 2019)**: Optimizes BERT's hyperparameters and training data, demonstrating that BERT was undertrained.
- **XLNet (Yang et al., 2020)**: Addresses BERT's independence assumption for predicted tokens by introducing permutation-based training.
- **BART (Lewis et al., 2019)**: Enhances the pre-training by corrupting the input texts in various ways and adding a reconstruction objective, essentially combining aspects of BERT and autoencoder architectures.
- **DeBERTa (He et al., 2021)**: Improves upon BERT by disentangling the word and position embeddings, providing a more refined understanding of word positions and context.
- **DistilBERT (Sanh et al., 2020)**: Offers a smaller, faster version of BERT that retains most of its performance, addressing the model's size and computational requirements.

Addressing Limitations and Bertology

While models like BERT have revolutionized NLP, they also come with limitations and areas for critical examination:

- **Bias and Ethics (Bender et al., 2021)**: Stochastic Parrot paper and other studies highlight the potential for biases in large language models and the ethical implications of their use.
- **Bertology (Rogers et al., 2020)**: A term coined to describe the extensive study of BERT's inner workings and behavior, aiming to demystify the model, understand its limitations, and improve its interpretability and fairness.
- **Model Efficiency**: Ongoing efforts to reduce the size and computational requirements of BERT-like models without significantly compromising performance.

Continuous Evolution: The field continues to evolve, with ongoing research addressing these challenges, improving model architectures, and ensuring that NLP technology progresses in a responsible and inclusive manner.

Different application of BERT-related models

SciBERT: A Pretrained Language Model for Scientific Text

Customization for the Scientific Domain:

- SciBERT (Beltagy et al., 2019) leverages the architecture of BERT but is trained on a corpus of scientific papers from Semantic Scholar.
- **Vocabulary Overlap**: Shares only 42% of its vocabulary with BERT, reflecting the unique terminology of scientific literature.
- **Training**: Follows the same configuration as BERT, ensuring robust learning from the scientific corpus.
- **Performance**: Achieves state-of-the-art results on 8 out of 12 scientific NLP tasks, outperforming BERT significantly (+2

Field	Task	Dataset	SOTA	BERT-Base		SciBERT	
				Frozen	Finetune	Frozen	Finetune
Bio	NER	BC5CDR (Li et al., 2016)	88.85[7]	85.08	86.72	88.73	**90.01**
		JNLPBA (Collier and Kim, 2004)	**78.58**	74.05	76.09	75.77	77.28
		NCBI-disease (Dogan et al., 2014)	**89.36**	84.06	86.88	86.39	88.57
	PICO	EBM-NLP (Nye et al., 2018)	66.30	61.44	71.53	68.30	**72.28**
	DEP	GENIA (Kim et al., 2003) - LAS	**91.92**	90.22	90.33	90.36	90.43
		GENIA (Kim et al., 2003) - UAS	**92.84**	91.84	91.89	92.00	91.99
	REL	ChemProt (Kringelum et al., 2016)	76.68	68.21	79.14	75.03	**83.64**
CS	NER	SciERC (Luan et al., 2018)	64.20	63.58	65.24	65.77	**67.57**
	REL	SciERC (Luan et al., 2018)	n/a	72.74	78.71	75.25	**79.97**
	CLS	ACL-ARC (Jurgens et al., 2018)	67.9	62.04	63.91	60.74	**70.98**
Multi	CLS	Paper Field	n/a	63.64	65.37	64.38	**65.71**
		SciCite (Cohan et al., 2019)	84.0	84.31	84.85	**85.42**	**85.49**
Average				73.58	77.16	76.01	79.27

Table 1: Test performances of all BERT variants on all tasks and datasets. **Bold** indicates the SOTA result (multiple results bolded if difference within 95% bootstrap confidence interval). Keeping with past work, we report macro F1 scores for NER (span-level), macro F1 scores for REL and CLS (sentence-level), and macro F1 for PICO (token-level), and micro F1 for ChemProt specifically. For DEP, we report labeled (LAS) and unlabeled (UAS) attachment scores (excluding punctuation) for the same model with hyperparameters tuned for LAS. All results are the average of multiple runs with different random seeds.

BERT vs. SciBERT Credit: Beltagy et al. (2019)

EconBERTa: Extraction of Named Entities in Economics

Advancing NER in Economic Research:

- **Objective**:NER in economics, specifically for extracting entities related to policy interventions from impact evaluation literature.
- **Context**: Addresses the lack of a dedicated dataset and model for NER in the economics domain, introducing the expert-annotated ECON-IE dataset.
- **Challenges Tackled**: Fills the gap in NER for economic impact evaluation by providing a robust model and a new dataset, addressing domain-specific extraction challenges.
- **Main Results**: SOA performance on the ECON-IE dataset, with insights into model generalization limitations.

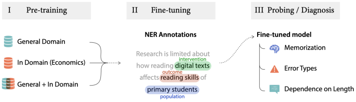


Figure 1: Illustration of the pipeline for the models under investigation, from modeling to diagnosis

Lasri et al. (2023)

XLM-T: RoBERTa Adapted for Twitter

Tailoring for Social Media - Twitter: Barbieri et al. (2021) adapted XLM-RoBERTa to analyze sentiment in tweets, spanning 30 languages.

- **Pre-training**: Dataset of 198M tweets, enhancing its understanding of social media language nuances.
- **Training**: Pre-training for 14 days on 8 NVIDIA V100 GPUs, focusing on Twitter's linguistic characteristics.
- **Fine-tuning**: Limited to the classification layer, maintaining the integrity of the pre-trained language understanding.
- **Results**: Demonstrates superior performance across languages in Twitter sentiment analysis compared to the base XLM-RoBERTa model.

	Monolingual			Bilingual		Multilingual	
	FT	**XLM-R**	**XLM-T**	**XLM-R**	**XLM-T**	**XLM-R**	**XLM-T**
Ar	45.98	63.56	**67.67**	63.63 (En)	67.65 (En)	64.31	66.89
En	50.85	68.18	66.89	65.07 (It)	67.47 (Es)	68.52	**70.63**
Fr	54.82	71.98	68.19	**73.55** (Sp)	68.24 (En)	70.52	71.18
De	59.56	73.61	76.13	72.48 (En)	75.49 (It)	72.84	**77.35**
Hi	37.08	36.60	40.29	33.57 (It)	55.35 (It)	53.39	**56.39**
It	54.65	71.47	70.91	70.43 (Ge)	**73.50** (Pt)	68.62	69.06
Pt	55.05	67.11	75.98	71.87 (Sp)	**76.08** (En)	69.79	75.42
Sp	50.06	65.87	68.52	67.68 (Po)	**68.68** (Pt)	66.03	67.91
All	51.01	64.80	66.82	64.78	69.06	66.75	**69.35**

Table 4: Cross-lingual sentiment analysis F1 results on target languages using target language training data (Monolingual) only, combined with training data from another language (Bilingual) and with all languages at once (Multilingual). "All" is computed as the average of all individual results.

Credit: Barbieri et al. (2021)

XtremeDistilTransformer: Lighter and Faster

Optimizing for Efficiency: Mukherjee et al. (2021) introduced XtremeDistilTransformer, focusing on distilling BERT's knowledge into a more compact model.

- **Distillation Process**: Condenses the information from a larger model into a smaller one without significant loss in performance.
- **Universality**: Utilizes task-specific techniques to maintain broad applicability.
- **Speed and Size**: 5 to 9 times faster inference speeds and a significantly reduced model size.
- **Performance**: Comparable or even superior to original larger models: an attractive choice for poc projects and applications with resource constraints.

Table 6: Comparing the performance of distilled models DistilBERT (Sanh, 2019), TinyBERT (Jiao et al., 2019), MiniLM (Wang et al., 2020) and XtremeDistilTransformers on the development set for several GLUE tasks. R denotes reported published results and HF denotes the performance obtained with our HuggingFace implementations.

Models	Params	Speedup	MNLI	QNLI	QQP	RTE	SST	MRPC	SQuADv2	Avg
BERT (R)	109	1x	84.5	91.7	91.3	68.6	93.2	87.3	76.8	84.8
BERT-Trun (R)	66	2x	81.2	87.9	90.4	65.5	90.8	82.7	69.9	81.2
DistilBERT (R)	66	2x	82.2	89.2	88.5	59.9	91.3	87.5	70.7	81.3
TinyBERT (R)	66	2x	83.5	90.5	90.6	72.2	91.6	88.4	73.1	84.3
MiniLM (R)	66	2x	84.0	91.0	91.0	71.5	92.0	88.4	76.4	84.9
MiniLM (R)	22	5.3x	82.8	90.3	90.6	68.9	91.3	86.6	72.9	83.3
BERT (HF)	109	1x	84.4	91.4	91.2	66.8	93.2	83.8	74.8	83.7
MiniLM (HF)	22	5.3x	82.7	89.4	90.3	64.3	90.8	84.1	71.5	81.9
XtremeDistilTransf. (HF)	**22**	**5.3x**	**84.5**	**90.2**	**90.4**	**77.3**	**91.6**	**89.0**	**74.4**	**85.3**
XtremeDistilTransf. (HF)	14	9.4x	81.8	86.9	89.5	74.4	89.9	86.5	63.0	81.7

Credit: Mukherjee et al. (2021)

ModernBERT: Enhancing BERT

Key Improvements over BERT:

- **Extended Context Length**: 8,192 tokens.
- **Rotary Positional Embeddings**: Improves handling of long sequences.
- **Flash Attention**: Enhances computational efficiency.
- **Bias-Free Linear Layers**: Reduces parameter count and potential overfitting.
- **Diverse Training Data**: Pretrained on 2 trillion tokens.
- Achieves **SOA** on benchmarks like GLUE, BEIR, and CodeSearchNet.
- Demonstrates faster inference and lower memory usage compared to BERT.

		IR (DPR)			IR (ColBERT)		NLU	Code	
	Model	BEIR	$MLDR_{OOD}$	$MLDR_{ID}$	BEIR	$MLDR_{OOD}$	GLUE	CSN	SQA
Base	BERT	38.9	23.9	32.2	49.0	28.1	84.7	41.2	59.5
	RoBERTa	37.7	22.9	32.8	48.7	28.2	86.4	44.3	59.6
	DeBERTaV3	20.2	5.4	13.4	47.1	21.9	88.1	17.5	18.6
	NomicBERT	41.0	26.7	30.3	49.9	61.3	84.0	41.6	61.4
	GTE-en-MLM	41.4	**34.3**	**44.4**	48.2	69.3	85.6	44.9	71.4
	ModernBERT	**41.6**	27.4	44.0	**51.3**	**80.2**	**88.4**	**56.4**	**73.6**
Large	BERT	38.9	23.3	31.7	49.5	28.5	85.2	41.6	60.8
	RoBERTa	41.4	22.6	36.1	49.8	28.8	88.9	47.3	68.1
	DeBERTaV3	25.6	7.1	19.2	46.7	23.0	**91.4**	21.2	19.7
	GTE-en-MLM	42.5	**36.4**	**48.9**	50.7	71.3	87.6	40.5	66.9
	ModernBERT	**44.0**	34.3	48.6	**52.4**	**80.4**	90.4	**59.5**	**83.9**

Credit: Warner et al. (2024)

QA and Takeaways

QA

Open Discussion

- Feel free to ask questions or share your thoughts about today's topics.
- Any insights, experiences, or perspectives you'd like to discuss are welcome.

Summary of Key Takeaways

- We explored the **attention mechanism**, addressing RNN limitations in handling long-range dependencies and enabling sequence processing parallelization.
- The **Transformer model** represents a paradigm shift, using self-attention for parallel processing and capturing intricate word interrelations without recurrent structures.
- **BERT** emerged as a pivotal NLP model, leveraging the Transformer's architecture for profound bidirectional context understanding, significantly advancing language task performance.
- Adaptations like **SciBERT**, **XLM-T**, and **XtremeDistilTransformer** demonstrate BERT's versatility, each pushing forward their respective domains.
- Acknowledged the models' **limitations and ethical considerations**, underscoring ongoing research needs in model efficiency, interpretability, and responsible AI development.