Machine Learning (CE 40477)

Fall 2024

Ali Sharifi-Zarchi

CE Department
Sharif University of Technology

November 23, 2024

Limitations of word2vec

- One vector for each word type
- Word2vec has challenges in polysemous words, e.g. `Light`
- Words don't appear in isolation. The word use (e.g., syntax and semantics) depends on its context.
- **Why not learn the representations for each word in its context?**

Contextualized Word Embeddings

- Build a vector for each word conditioned on its **context**.

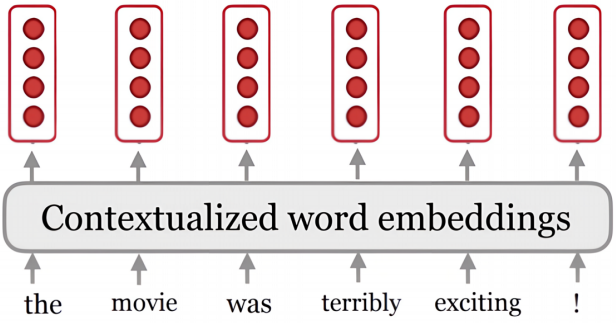


Figure 1: representation for each token is a function of the entire input sentence

Contextualized Word Embeddings

Compute contextual vector:

$$\mathbf{c_k} = f(w_k \mid w_1, w_2, \ldots, w_n) \in \mathbb{R}^d$$

Examples:

$f(light \mid \text{Please turn off the } \overbrace{light}^{} .)$

$\neq$

$f(light \mid \text{This box is very } \underbrace{light}_{} \text{ to carry.})$

How do we implement the context function f?

Enter Recurrent Neural Networks

- RNNs: Natural choice for contextual representations.
- Process words sequentially, maintaining context.
- Each hidden state captures information about:
  - Current word
  - All previous words in sequence

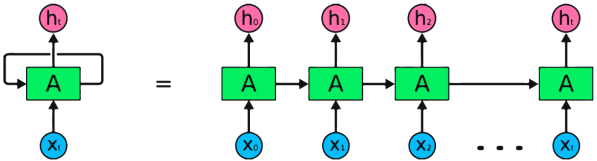


Figure 2: where h_t can serve as our contextualized representation.

RNN hidden state update

- Process sequences by maintaining a hidden state.
- Updates state sequentially: $h_t = f_W(h_{t-1}, x_t)$
- This recurrence formula is applied at each time step to process a sequence of vectors x.
- The same function and the same set of parameters are used at every time step.

$$h_t = f_W(h_{t-1}, x_t)$$

new state

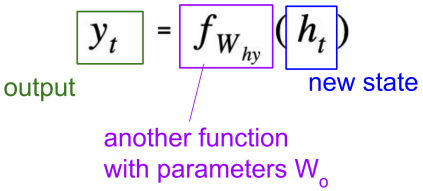
old state

input vector at some time step

some function with parameters W

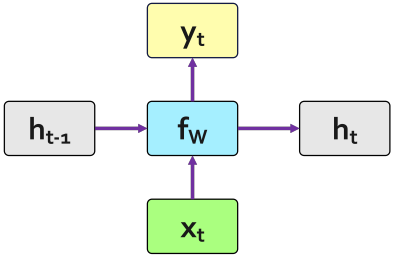
RNN Output Generation

- After updating the hidden state, RNNs generate outputs at each time step.
- The output y_t is typically computed as a function of the current hidden state h_t, often passed through a layer (like a softmax layer) for classification or regression tasks.



(Vanilla) Recurrent Neural Network

- The state of the RNN consists of a single "hidden" vector h_t, which updates as new inputs are processed.



$$h_t = f_W(h_{t-1}, x_t)$$

$$\downarrow$$

$$h_t = f(W_{hh}h_{t-1} + W_{xh}x_t + b)$$

$$y_t = W_{hy}h_t$$

Challenges of RNNs

- While improved versions of vanilla RNNs attempt to resolve some issues, they still struggle with:
  - Long-term dependencies
  - Sequential computation (can't parallelize)
- **So, what solutions do we have?**

Attention is all you need!

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 41.0 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature.

https://arxiv.org/abs/1706.03762

Why Attention?

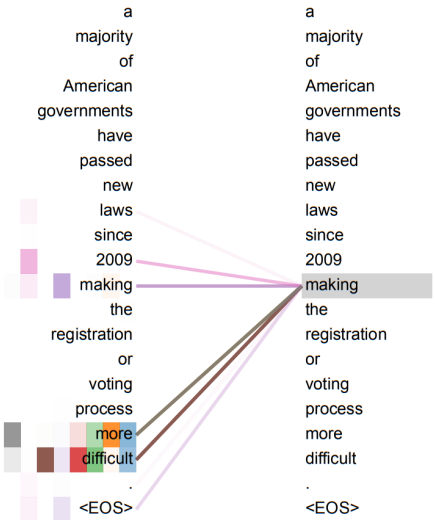
- Enhances the model's ability to focus on relevant parts of the input.
- Can compute relationships between inputs regardless of their position.
- Inspired by human visual attention.
- **Key Idea**
  - Let the model learn which parts of the input are important for each output.

Motivation for Attention in RNNs

Attention Mechanism

- Helps RNNs focus on important parts of the sequence.
- Allows model to “attend” the most relevant information from the entire sequence, rather than relying solely on the hidden state at the current time step.
- Improves the model’s ability to handle long-range dependencies by:
  - Directly accessing key information at any position in the input sequence.
  - Avoiding the need to compress all information into a single hidden state.

How Attention Works?



Attention as a Soft, Averaging Lookup Table

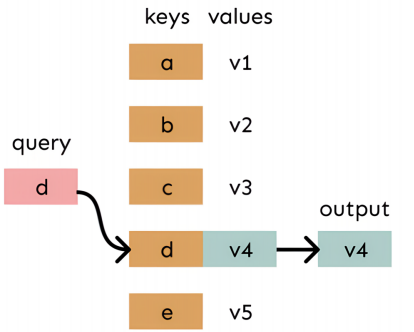


Figure 4: In a lookup table, we have a table of keys that map to values. The query matches one of the keys, returning its value.

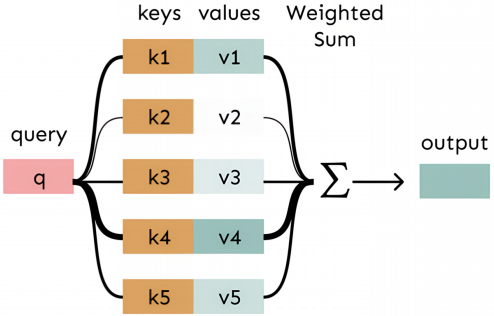


Figure 5: In the attention, the query matches all keys softly, to a weight between 0 and 1. The keys' values are multiplied and summed by the weights.

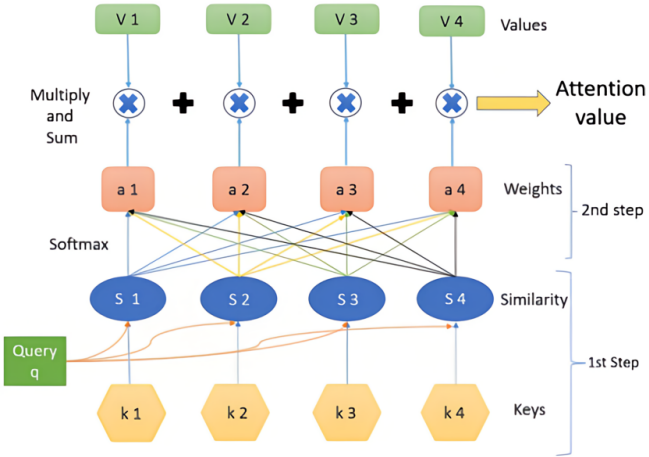
Attention Mechanism: Basic Formula

- Query (Q): What we're looking for
- Key (K): What we match against
- Value (V): What we retrieve

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

- d_k: dimension of keys (scaling factor)
- Softmax converts scores to probabilities

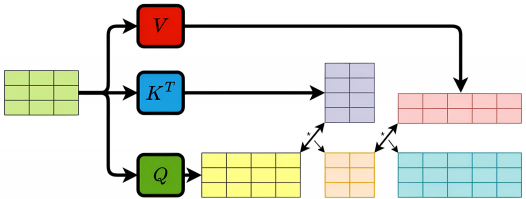
Attention Mechanism: Basic Formula



Self-Attention vs Cross-Attention

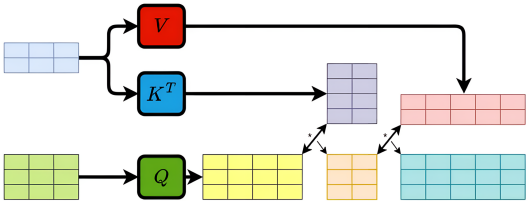
Self-Attention

- Q, K, V from the same sequence
- Each position attends to all positions
- Used in encoder and decoder



Cross-Attention

- Q from one sequence
- K, V from another
- Used in encoder-decoder attention



Self-Attention: Core Theorem

Self-Attention Properties

Self-attention layers can model dependencies between all elements in an input sequence in parallel, capturing long-range relationships efficiently.

- Global connectivity
- Parallel computation
- Position-independent weighting
- $O(n^2)$ complexity for sequence length n

Self-Attention: Mathematical Foundation

- Input sequence: $X = [x_1, x_2, ..., x_n] \in \mathbb{R}^{n \times d}$
- Linear projections:

$$Q = XW^Q \in \mathbb{R}^{n \times d_k}$$
$$K = XW^K \in \mathbb{R}^{n \times d_k}$$
$$V = XW^V \in \mathbb{R}^{n \times d_v}$$

- Attention computation:

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

Proof Component 1: Parallel Processing

- Matrix multiplication enables parallel computation:

$$S = QK^T = \begin{bmatrix} (q_1 k_1^T) & \cdots & (q_1 k_n^T) \\ \vdots & \ddots & \vdots \\ (q_n k_1^T) & \cdots & (q_n k_n^T) \end{bmatrix}$$

- All n^2 interactions computed simultaneously
- No sequential dependencies between computations

Key Insight

Unlike RNNs, there is no need to wait for previous timesteps

Proof Component 2: Global Dependencies

- Attention weights between positions i and j:

$$\alpha_{ij} = \frac{\exp(q_i^T k_j / \sqrt{d_k})}{\sum_{l=1}^{n} \exp(q_i^T k_l / \sqrt{d_k})}$$

  - Properties:
  - α_{ij} depends only on compatibility of i and j
  - No distance-based attenuation
  - Softmax ensures $\sum_j \alpha_{ij} = 1$

Proof Component 2: Global Dependencies

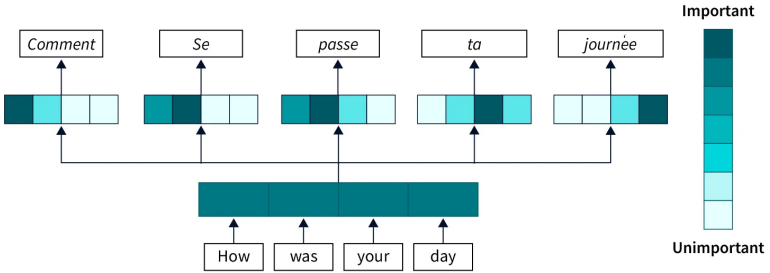


Figure 6: Attention weight visualization

Proof Component 3: Computational Efficiency

Complexity Analysis

- Matrix multiply: $O(n^2 d)$
- Softmax: $O(n^2)$
- Total: $O(n^2 d)$

Optimizations

- Sparse attention
- Linear attention
- Sliding window

Memory-Computation Tradeoff

More memory than RNNs ($O(n)$), but enables parallelization.

Implementation Details

- Practical considerations:
  - Scale factor $\sqrt{d_k}$ prevents vanishing gradients
  - Mask for causal attention (decoder)
  - Dropout for regularization

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

where $M_{ij} = -\infty$ for masked positions

Formal Proof Conclusion

Theorem Verification

Self-attention satisfies all claimed properties:

1. Parallel computation: Matrix operations
2. Global dependencies: Direct pairwise attention
3. Efficient computation: $O(n^2 d)$ complexity
4. Learnable patterns: Attention weights

Important Note

These properties make self-attention particularly suitable for:

- Natural language processing
- Graph neural networks
- Computer vision (with modifications)

Positional Encoding

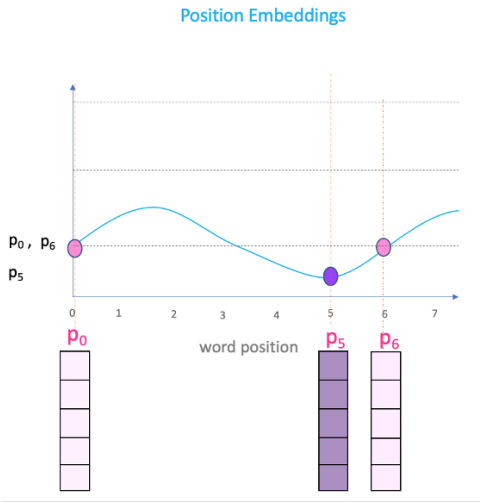
- Problem: Attention is position-agnostic
- Solution: Add position information to embeddings
- Using sinusoidal functions:

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}})$$

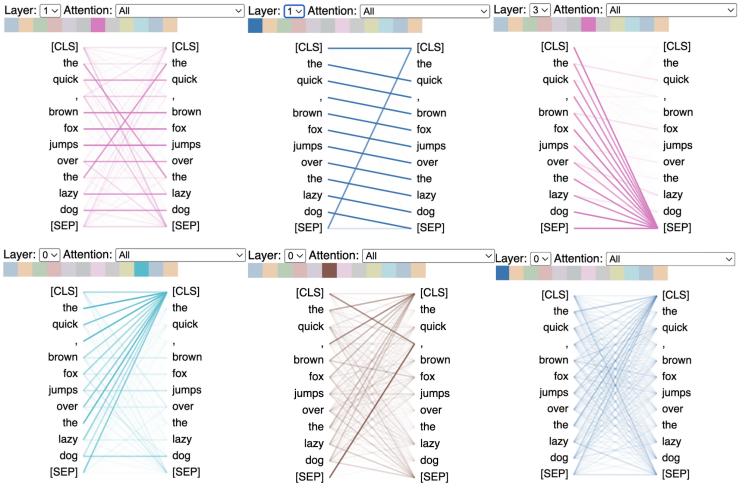
$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}})$$

- Allows model to learn relative positions
- Works for sequences of any length

Positional Encoding



Multi-Head Attention



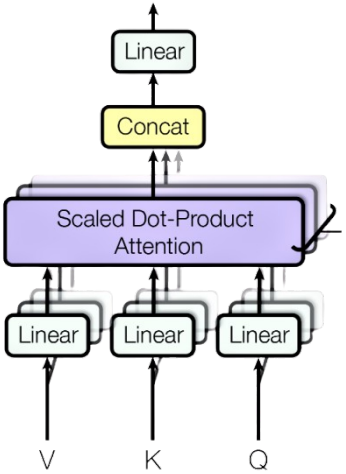
Multi-Head Attention

- Instead of single attention, use multiple heads
- Each head can focus on different aspects
- Process in parallel and concatenate

$$\text{MultiHead}(Q, K, V) = \text{Concat}(head_1, \ldots, head_h)\, W^O$$

$$head_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Multi-Head Attention



Multi-Head Attention Benefits

- Multiple representation subspaces
- Can capture different types of relationships:
  - Syntactic dependencies
  - Semantic relationships
  - Long-range dependencies
- Improves model capacity and stability

Contributions

These slides are authored by:

- Faezeh Sarlakifar

[1] F.-F. Li and Z. Durante, *CS231n Lectures*.
Stanford University, June 2024.
Updated June 3, 2024.

[2] C. Manning, *CS224n Lectures*.
Stanford University, June 2024.

[3] M. Soleymani Baghshah, "Deep learning." Lecture slides.

Any Questions?