Attention in Neural Networks

Nina Poerner

December 5, 2018

Why attention?

- Limitation of CNN
  - Can only capture local dependencies

Why attention?

- Limitation of CNN
  - Can only capture local dependencies
- Limitations of LSTM, GRU, etc.:
  - Can capture long-range dependencies, but may find them difficult to learn
  - Sequential processing not parallelizable

Why attention?

- Limitation of CNN
  - Can only capture local dependencies
- Limitations of LSTM, GRU, etc.:
  - Can capture long-range dependencies, but may find them difficult to learn
  - Sequential processing not parallelizable
  - In Sequence2Sequence (Encoder-Decoder) architectures: Must fit all source sentence info into a single fixed-size vector
    - Fails when source sentence is very long
    - Major issue in early NMT architectures
    - Attention was first proposed for Sequence2Sequence / NMT (Bahdanau et al. 2015)

Reading Group: Bahdanau et al.

Attention with one query vector

- Query: $\mathbf{q} \in \mathbb{R}^{D^q}$
  - vector of size D^q, Bahdanau: $\mathbf{q} = \mathbf{s}_{i-1}$
- Keys: $\mathbf{K} \in \mathbb{R}^{T \times D^k}$
  - matrix of size $T \times D^k$, Bahdanau: $\mathbf{K} = \mathbf{h}_1 \dots \mathbf{h}_{T^x}$
- Values: $\mathbf{V} \in \mathbb{R}^{T \times D^v}$
  - matrix of size $T \times D^v$, Bahdanau: $\mathbf{V} = \mathbf{h}_1 \dots \mathbf{h}_{T^x}$

Attention with one query vector

- Query: $\mathbf{q} \in \mathbb{R}^{D^q}$
  - vector of size D^q, Bahdanau: $\mathbf{q} = \mathbf{s}_{i-1}$
- Keys: $\mathbf{K} \in \mathbb{R}^{T \times D^k}$
  - matrix of size $T \times D^k$, Bahdanau: $\mathbf{K} = \mathbf{h}_1 \dots \mathbf{h}_{T^x}$
- Values: $\mathbf{V} \in \mathbb{R}^{T \times D^v}$
  - matrix of size $T \times D^v$, Bahdanau: $\mathbf{V} = \mathbf{h}_1 \dots \mathbf{h}_{T^x}$
- Energy Function: $\mathbf{e} = f(\mathbf{q}, \mathbf{K}); \mathbf{e} \in \mathbb{R}^T$
  - vector of size T
  - form of f varies, e.g.
    - ⋆ dot product: $\mathbf{e} = \mathbf{q}\mathbf{K}^T$
    - ⋆ Bahdanau: multilayer perceptron

Attention with one query vector

- Query: $\mathbf{q} \in \mathbb{R}^{D^q}$
  - vector of size D^q, Bahdanau: $\mathbf{q} = \mathbf{s}_{i-1}$
- Keys: $\mathbf{K} \in \mathbb{R}^{T \times D^k}$
  - matrix of size $T \times D^k$, Bahdanau: $\mathbf{K} = \mathbf{h}_1 \ldots \mathbf{h}_{T^x}$
- Values: $\mathbf{V} \in \mathbb{R}^{T \times D^v}$
  - matrix of size $T \times D^v$, Bahdanau: $\mathbf{V} = \mathbf{h}_1 \ldots \mathbf{h}_{T^x}$
- Energy Function: $\mathbf{e} = f(\mathbf{q}, \mathbf{K}); \mathbf{e} \in \mathbb{R}^T$
  - vector of size T
  - form of f varies, e.g.
    - ★ dot product: $\mathbf{e} = \mathbf{q}\mathbf{K}^T$
    - ★ Bahdanau: multilayer perceptron
- Attention: $\mathbf{a} = \mathrm{softmax}(\mathbf{e})$
  - probability distribution over T

Attention with one query vector

- Query: $\mathbf{q} \in \mathbb{R}^{D^q}$
  - vector of size D^q, Bahdanau: $\mathbf{q} = \mathbf{s}_{i-1}$
- Keys: $\mathbf{K} \in \mathbb{R}^{T \times D^k}$
  - matrix of size $T \times D^k$, Bahdanau: $\mathbf{K} = \mathbf{h}_1 \dots \mathbf{h}_{T^x}$
- Values: $\mathbf{V} \in \mathbb{R}^{T \times D^v}$
  - matrix of size $T \times D^v$, Bahdanau: $\mathbf{V} = \mathbf{h}_1 \dots \mathbf{h}_{T^x}$
- Energy Function: $\mathbf{e} = f(\mathbf{q}, \mathbf{K}); \mathbf{e} \in \mathbb{R}^T$
  - vector of size T
  - form of f varies, e.g.
    - dot product: $\mathbf{e} = \mathbf{q}\mathbf{K}^T$
    - Bahdanau: multilayer perceptron
- Attention: $\mathbf{a} = \mathrm{softmax}(\mathbf{e})$
  - probability distribution over T
- Output: $\mathbf{o} = \mathbf{a}\mathbf{V}; \mathbf{o} \in \mathbb{R}^{D^v}$
  - sum of the T value vectors weighted by attention

Generalization 1: Multiple query vectors

- Compute energy, attention and output vectors for multiple query vectors (query matrix) **in parallel!**
- Query: $\mathbf{Q} \in \mathbb{R}^{T^q \times D^q}$
  - matrix of size $T^q \times D^q$
- Keys: $\mathbf{K} \in \mathbb{R}^{T^v \times D^k}$
  - matrix of size $T^v \times D^k$
- Values: $\mathbf{V} \in \mathbb{R}^{T^v \times D^v}$
  - matrix of size $T^v \times D^v$
  - T^q may be different from T^v, but the number of values must be equal to number of keys (why?)

Generalization 1: Multiple query vectors

- Compute energy, attention and output vectors for multiple query vectors (query matrix) **in parallel!**
- Query: $\mathbf{Q} \in \mathbb{R}^{T^q \times D^q}$
  - matrix of size $T^q \times D^q$
- Keys: $\mathbf{K} \in \mathbb{R}^{T^v \times D^k}$
  - matrix of size $T^v \times D^k$
- Values: $\mathbf{V} \in \mathbb{R}^{T^v \times D^v}$
  - matrix of size $T^v \times D^v$
  - T^q may be different from T^v, but the number of values must be equal to number of keys (why?)
- Energy Function: $\mathbf{E} = f(\mathbf{Q}, \mathbf{K}); \mathbf{E} \in \mathbb{R}^{T^q \times T^v}$
  - matrix of size $T^q \times T^v$

Generalization 1: Multiple query vectors

- Compute energy, attention and output vectors for multiple query vectors (query matrix) **in parallel!**
- Query: $\mathbf{Q} \in \mathbb{R}^{T^q \times D^q}$
  - matrix of size $T^q \times D^q$
- Keys: $\mathbf{K} \in \mathbb{R}^{T^v \times D^k}$
  - matrix of size $T^v \times D^k$
- Values: $\mathbf{V} \in \mathbb{R}^{T^v \times D^v}$
  - matrix of size $T^v \times D^v$
  - T^q may be different from T^v, but the number of values must be equal to number of keys (why?)
- Energy Function: $\mathbf{E} = f(\mathbf{Q}, \mathbf{K}); \mathbf{E} \in \mathbb{R}^{T^q \times T^v}$
  - matrix of size $T^q \times T^v$
- Attention: $\mathbf{A} = \mathrm{softmax}(\mathbf{E})$
  - Softmax over T^v dimension
  - For every query, one probability distribution over T^v

Generalization 1: Multiple query vectors

- Compute energy, attention and output vectors for multiple query vectors (query matrix) **in parallel!**
- Query: $\mathbf{Q} \in \mathbb{R}^{T^q \times D^q}$
  - matrix of size $T^q \times D^q$
- Keys: $\mathbf{K} \in \mathbb{R}^{T^v \times D^k}$
  - matrix of size $T^v \times D^k$
- Values: $\mathbf{V} \in \mathbb{R}^{T^v \times D^v}$
  - matrix of size $T^v \times D^v$
  - T^q may be different from T^v, but the number of values must be equal to number of keys (why?)
- Energy Function: $\mathbf{E} = f(\mathbf{Q}, \mathbf{K}); \mathbf{E} \in \mathbb{R}^{T^q \times T^v}$
  - matrix of size $T^q \times T^v$
- Attention: $\mathbf{A} = \mathrm{softmax}(\mathbf{E})$
  - Softmax over T^v dimension
  - For every query, one probability distribution over T^v
- Output: $\mathbf{O} = \mathbf{AV}; \mathbf{O} \in \mathbb{R}^{T^q \times D^v}$
  - For every query, one weighted sum over the value vectors

Generalization 2: Multi-head attention

- Given: Query, key and value matrices $\mathbf{Q}$, $\mathbf{K}$, $\mathbf{V}$

Generalization 2: Multi-head attention

- Given: Query, key and value matrices $\mathbf{Q}$, $\mathbf{K}$, $\mathbf{V}$
- Define $3n$ fully connected layers
- Generate n sets of queries, n sets of keys and n sets of values
  - $\mathbf{Q}^1 = \mathbf{Q}\mathbf{W}^{Q1}, \ldots, \mathbf{Q}^n = \mathbf{Q}\mathbf{W}^{Qn}$
  - $\mathbf{K}^1 = \mathbf{K}\mathbf{W}^{K1}, \ldots, \mathbf{K}^n = \mathbf{K}\mathbf{W}^{Kn}$
  - $\mathbf{V}^1 = \mathbf{V}\mathbf{W}^{V1}, \ldots, \mathbf{V}^n = \mathbf{V}\mathbf{W}^{Vn}$

Generalization 2: Multi-head attention

- Given: Query, key and value matrices $\mathbf{Q}$, $\mathbf{K}$, $\mathbf{V}$
- Define $3n$ fully connected layers
- Generate n sets of queries, n sets of keys and n sets of values
  - $\mathbf{Q}^1 = \mathbf{Q}\mathbf{W}^{Q1}, \ldots, \mathbf{Q}^n = \mathbf{Q}\mathbf{W}^{Qn}$
  - $\mathbf{K}^1 = \mathbf{K}\mathbf{W}^{K1}, \ldots, \mathbf{K}^n = \mathbf{K}\mathbf{W}^{Kn}$
  - $\mathbf{V}^1 = \mathbf{V}\mathbf{W}^{V1}, \ldots, \mathbf{V}^n = \mathbf{V}\mathbf{W}^{Vn}$
- Apply attention to every triple: $\mathbf{O}^1, \ldots, \mathbf{O}^n$
- Concatenate outputs along last axis: $\mathbf{O} = [\mathbf{O}^1 || \ldots || \mathbf{O}^n]$

Generalization 2: Multi-head attention

- Given: Query, key and value matrices $\mathbf{Q}$, $\mathbf{K}$, $\mathbf{V}$
- Define $3n$ fully connected layers
- Generate n sets of queries, n sets of keys and n sets of values
  - $\mathbf{Q}^1 = \mathbf{Q}\mathbf{W}^{Q1}, \ldots, \mathbf{Q}^n = \mathbf{Q}\mathbf{W}^{Qn}$
  - $\mathbf{K}^1 = \mathbf{K}\mathbf{W}^{K1}, \ldots, \mathbf{K}^n = \mathbf{K}\mathbf{W}^{Kn}$
  - $\mathbf{V}^1 = \mathbf{V}\mathbf{W}^{V1}, \ldots, \mathbf{V}^n = \mathbf{V}\mathbf{W}^{Vn}$
- Apply attention to every triple: $\mathbf{O}^1, \ldots, \mathbf{O}^n$
- Concatenate outputs along last axis: $\mathbf{O} = [\mathbf{O}^1 || \ldots || \mathbf{O}^n]$
- Intuition: Different heads focus on different dependencies
- Extremely parallelizable (every query in every head)

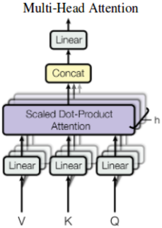

Image: Vaswani et al: Attention is all you need (2017, NIPS)

Self-Attention

- Queries, keys and values are derived from the same sequence of vectors
- E.g., given a sequence of hidden vectors $\mathbf{H} = \mathbf{h}_1 \dots \mathbf{h}_T$
  - $\mathbf{Q} = \mathbf{H}\mathbf{W}^Q$
  - $\mathbf{K} = \mathbf{H}\mathbf{W}^K$
  - $\mathbf{V} = \mathbf{H}\mathbf{W}^V$
- The sequence attends to itself!
- Can be built on top of an RNN, CNN
- ... or can even replace RNN, CNN completely (see next slice)

Attention is all you need?

- Transformer architecture: NMT architecture that is built with just attention, no RNNs, CNNs
- Can be used for Sequence2Sequence, but also language modeling, text classification (as self-attention)

Attention is all you need?

- Problem 1: Transformer has no sense of relative or absolute positions (why?)
- Solution: Add position embeddings to word embeddings!
  - Lookup table P of size (maxlen $\times$ embedding dim)
  - First word mapped to p_1, 7th word mapped to p_7, etc.
  - Trainable or deterministic (sinusoid embeddings)

Attention is all you need?

- Problem 1: Transformer has no sense of relative or absolute positions (why?)
- Solution: Add position embeddings to word embeddings!
  - Lookup table P of size (maxlen $\times$ embedding dim)
  - First word mapped to p_1, 7th word mapped to p_7, etc.
  - Trainable or deterministic (sinusoid embeddings)
- Problem 2: Transformer must be deep in order to work $\rightarrow$ gradients may explode
- Solution 2: normalize activations after each layer

Attention is all you need?

- Problem 1: Transformer has no sense of relative or absolute positions (why?)
- Solution: Add position embeddings to word embeddings!
  - Lookup table P of size (maxlen $\times$ embedding dim)
  - First word mapped to p_1, 7th word mapped to p_7, etc.
  - Trainable or deterministic (sinusoid embeddings)
- Problem 2: Transformer must be deep in order to work $\rightarrow$ gradients may explode
- Solution 2: normalize activations after each layer
- Problem 3: When decoding / language modeling, how can we keep the model from attending to future input words?
- Solution 3: Set $e_{ij} = -\inf$ where $j > i$ ("masked attention")

Attention is all you need?

- Problem 1: Transformer has no sense of relative or absolute positions (why?)
- Solution: Add position embeddings to word embeddings!
  - Lookup table P of size (maxlen $\times$ embedding dim)
  - First word mapped to p_1, 7th word mapped to p_7, etc.
  - Trainable or deterministic (sinusoid embeddings)
- Problem 2: Transformer must be deep in order to work $\rightarrow$ gradients may explode
- Solution 2: normalize activations after each layer
- Problem 3: When decoding / language modeling, how can we keep the model from attending to future input words?
- Solution 3: Set $e_{ij} = -\inf$ where $j > i$ ("masked attention")
- Can be difficult to train, sensitive to learning rate (Chen et al.)

Attention is all you need

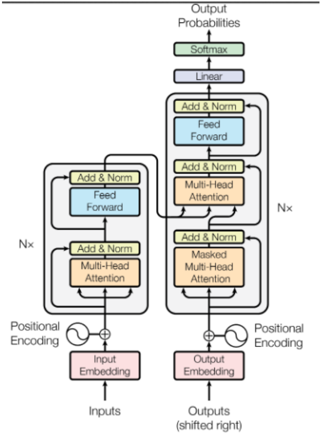

Figure 1: The Transformer - model architecture.