Attention and Transformers

Gautam Kamath

Converting Sentences into Matrices?

- “The cat sat on the mat”
- How to convert to vectors?
- Naïve: 1-hot encoding
  - the -> [1, 0, 0, …]
  - cat -> [0, 1, 0, …]
  - Sat -> [0, 0, 1, …]
- Downsides:
  - Very high dimensional (dimension = size of vocabulary > 1 million?)
    - 50,000 words cover 97% of text, but remaining 3% is important! (example on board)
  - Semantic meaning of vectors? (example on board)

Word Embeddings

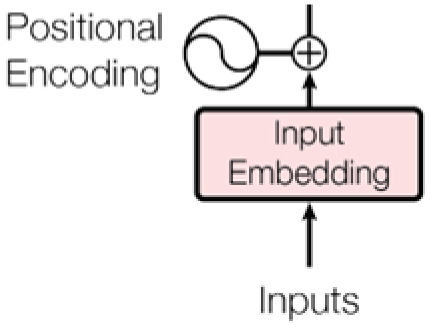
- Idea: “The cat ??? on the” – what do you think ??? is?
- Map from one-hot to embeddings using W, average them, map back to one-hot using W', softmax and cross-entropy loss (draw)
  - Continuous bag of words, CBOW
  - Also Skip-Gram (draw briefly)
- Train, then keep W
- King – Man + Woman = Queen (draw)
- Problems with word embeddings?
  - Polysemy ("I went to the bank to deposit money" versus "We sat on the river bank.“)
  - **What about words we’ve never seen before?**

Tokenization

- Suppose “unreachable” never appears in the training data
  - Un-reach-able
- Tokenization: split words into sub-words, aka tokens
- How to tokenize?
  - Many algorithms, Byte Pair Encoding (BPE), WordPiece, Unigram LM
- General principle: start with 1-hot, map down with learned W
- Still some problems
  - Strawberry
  - Still doesn’t solve polysemy…

Positional Encoding

- “The man ate the fish” versus “The fish ate the man”
- Positional encodings store location in sentence
- Input at position t = word embedding for word at position t + positional encoding for position t
  - $PE_t(2i) \approx \sin\left(\frac{t}{10000^{2i/d}}\right)$, use cosine for odd indices



Sequence Modelling

- Suppose we have a sequence of length n, each element in the sequence is of dimension d
- Previous solution: RNNs
  - Computing each hidden state is $O(d^2)$, so overall $O(nd^2)$ computation
  - Long chains make it hard to deal with long-range dependencies
  - Optimization woes like vanishing and exploding gradients
  - Many training steps
  - Sequential nature makes it hard to parallelize
- The attention mechanism solves most of these
  - …though computation increases for long sequences

Attention Mechanism/Layer

- Takes three inputs: set of queries q_j, keys k_i, and values v_i
  - Same number of keys and values, number of queries may differ
- For a given query q_j, tries to mimic retrieval lookup of value v_i corresponding to key k_i which "matches" query
  - Hard retrieval: SELECT value FROM table WHERE key = query
  - Soft retrieval: SELECT weighted_sum(values) FROM table WHERE key ≈ query
- $\text{attention}\left(q_j, \{k_i\}, \{v_i\}\right) = \sum_i \text{similarity}\left(q_j, k_i\right) \times v_i$
  - Softmax similarities to get a weighted sum
- (Draw: layer 1 is q_j and $k_i \rightarrow s_i$, layer 2 is softmax to get a_i's, layer 3 is multiplication and sum with a_i and v_i to produce output)

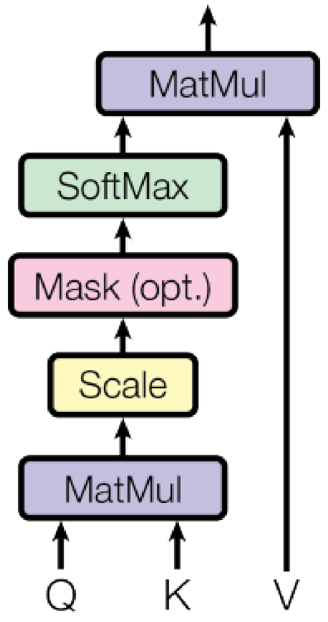
Similarity?

- What is $\mathrm{similarity}(q, k)$?
- Dot product: $\langle q, k \rangle$
  - q and k are of dimension d – what if they're mean 0 and variance 1 each?
- Scaled dot product: $\frac{\langle q, k \rangle}{\sqrt{d}}$
  - Plays nicer for softmax
- General dot product: $\langle Aq, Bk \rangle$
  - A and B are matrices of learnable parameters

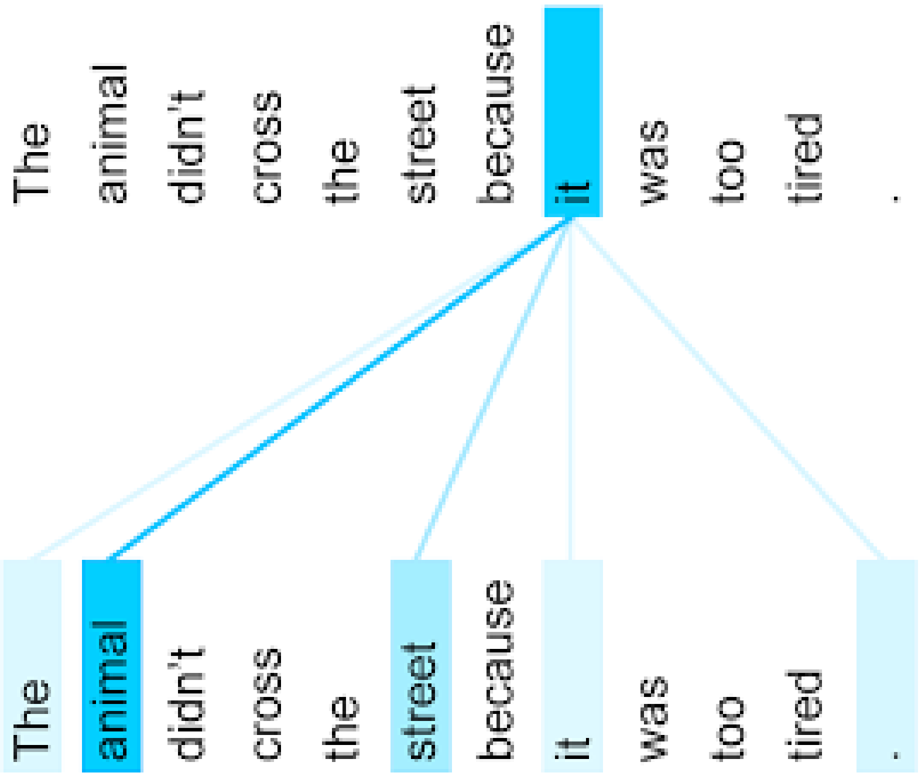
Attention for Sets

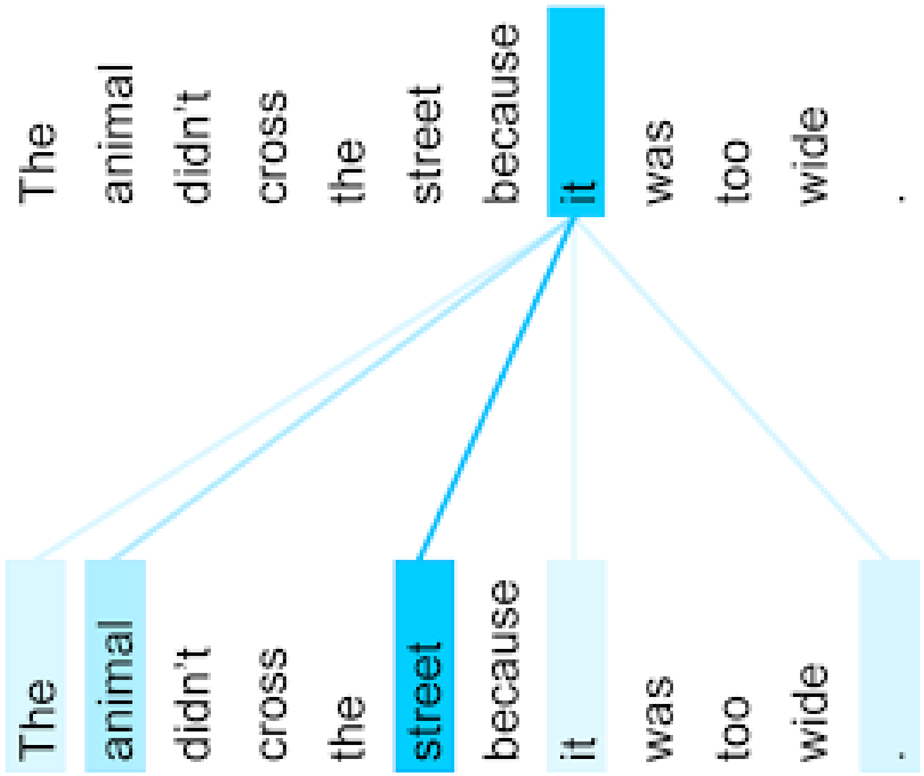
- Since similarity between vectors q and k is $\langle q, k \rangle$, how do we compute similarity between *sets* of vectors Q and K?
- Matrix multiplication: QK^T
- Attention mechanism looks like →
- Self-attention: when $Q = K = V$
- (Draw example with “word” matrices attn)
- $\mathrm{softmax}(VV^T)$: each row becomes distribution
- $\mathrm{softmax}(VV^T)V$: replace rows with weighted sums

Scaled Dot-Product Attention



Visualizing Attention



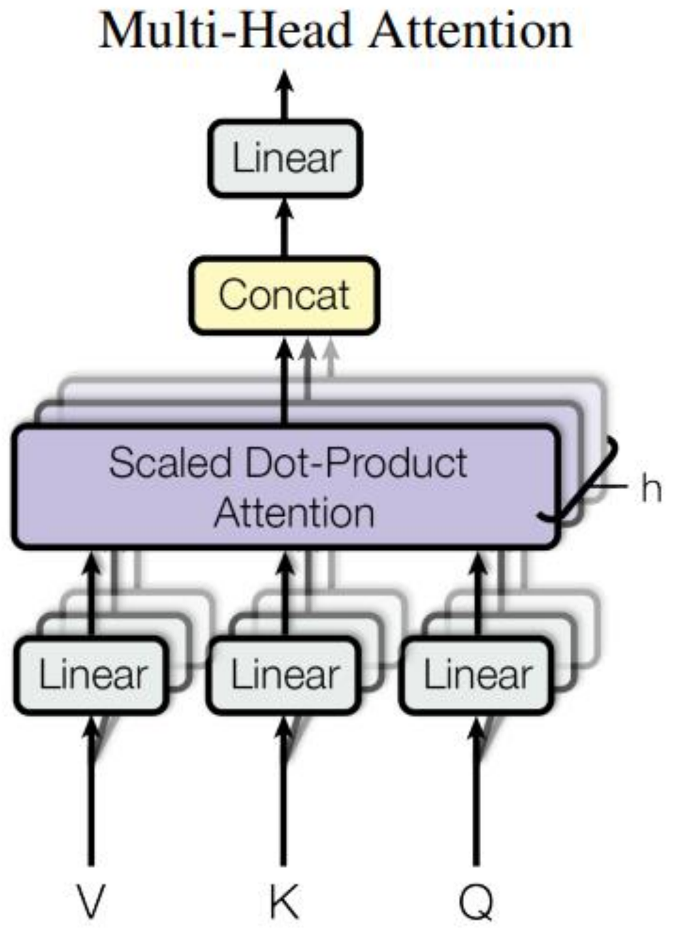


Attention vs RNN for sequences

- Sequence of n vectors, each d-dimensional
- Computation: $O(n^2 d)$
  - VV^T computation is $O(n^2 d)$
- Compare with RNNs: $O(nd^2)$
- Advantage of attention: maximum sequence length is $O(1)$
  - No long dependence chains for long sequences

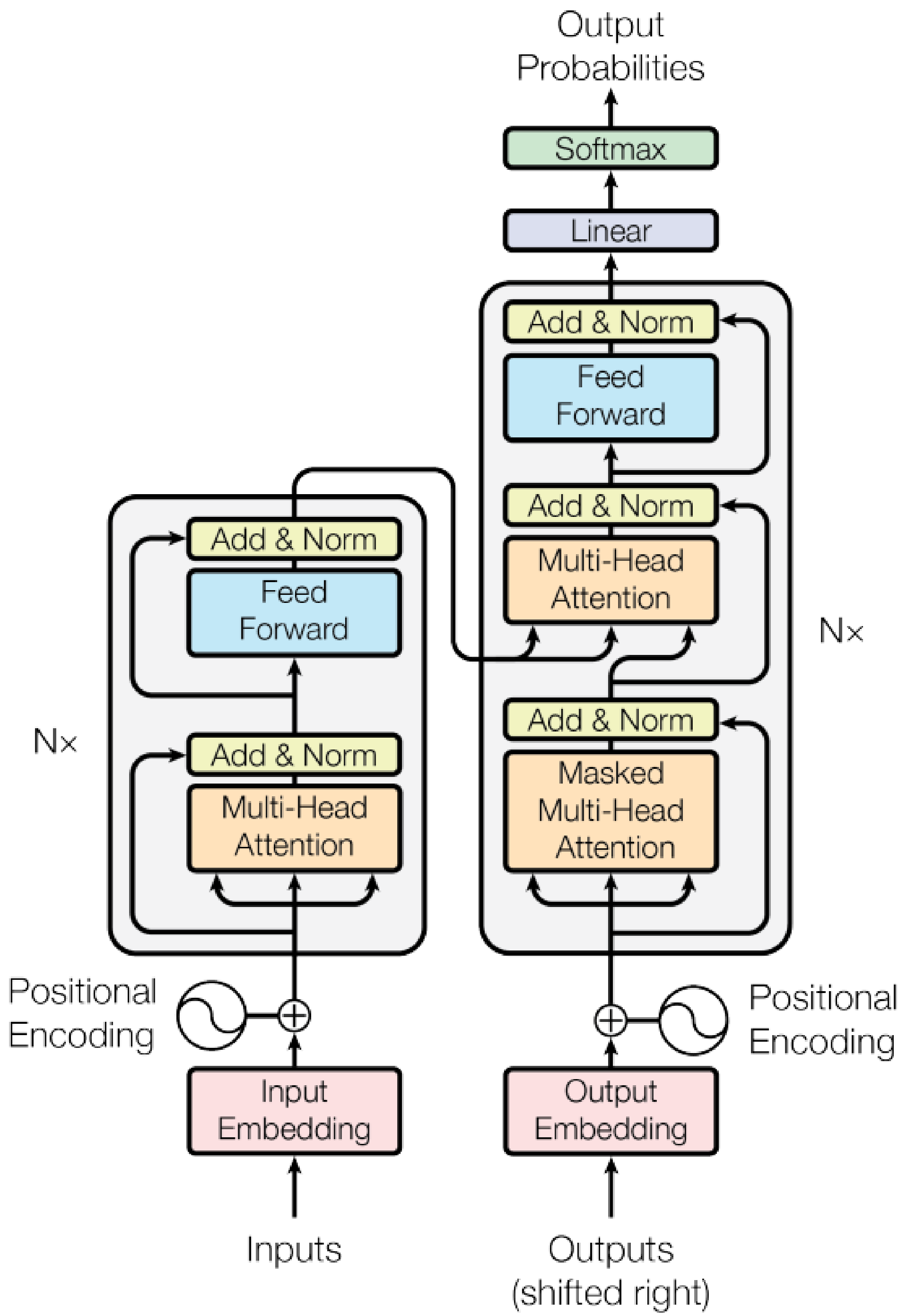
Multi-head attention

- Precede Attention with linear layer
  - General dot product
- Multiple “heads” in parallel
  - Similar to multiple filters in CNNs



Transformer Architecture

- Encoder-decoder structure
- Other layers:
- Input embedding
- Positional encoding
- Layer normalization
  - For each vector, normalize entries



Masked (Self-)Attention

- Say you're doing next word prediction… prevent "cheating" by looking at the next word!
- Self-attention: $\mathrm{softmax}(VV^T)V$
- Masked self-attention: $\mathrm{softmax}(\mathrm{mask}(VV^T))V$
- $mask(M)_{ij} = -\infty$ if $i < j$, M_{ij} otherwise (draw mask)

Results

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

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**	$2.3 \cdot 10^{19}$	