TRANSFORMERS

CMPT 728/420

Introduction to Deep Learning

Oliver Schulte

OVERVIEW

- Transformers are a state-of-the-art sequence-to-sequence model.
  - Used in [many state-of-the-art NLP systems]
  - E.g. BERT word embeddings (see bonus question on assignment)
  - ChatGPT = Chat Generative Pre-trained **Transformer**
  - Demos: https://transformer.huggingface.co
- They transform
  - one sequence into another
  - Initial generic word embeddings to context-dependent word embeddings
- Many moving parts
  - And many hyperparameters
- We will focus on the fundamental new ideas

HIGH-LEVEL IDEA

- Transformers use a **stateless model**
  - No (hidden) state summarizing previous observations
- Complex but separate embedding of each input item at time t
- When predicting output at target time t, access all embeddings at other times in random-access manner
  - Attention weights for how important embeddings for other times are for the current target time
- Insight: Natural language aims to convey a *set* of facts
  - ➢ *order* of words is not so important

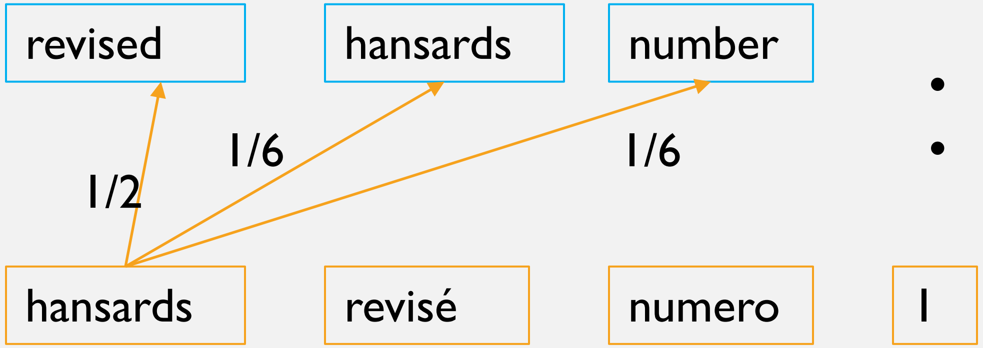
“In Korea, more than half the residents speak Korean”

REVIEW: ATTENTION IN SEQ-2-SEQ MODELS

REVIEW: ATTENTION IN SEQ2SEQ

- Basic Positional Attention Model for encoder-decoder RNNs
- Each decoder step accesses each hidden state of the encoder.
- The relevance of an input position to an input position is represented by an attention weight.
- Visualization

TOY EXAMPLE



- Matching positions have higher weights
- Requires fixed windows to fix positions

Input/Output	1	2	3
1	1/2	1/6	1/6
2	1/6	1/3	1/6
3	1/6	1/6	1/3
4	1/6	1/6	1/3

SELF-ATTENTION

["Attention is All You Need"]

Single-Sequence Attention

REVIEW: SINGLE-SEQUENCE MODELS

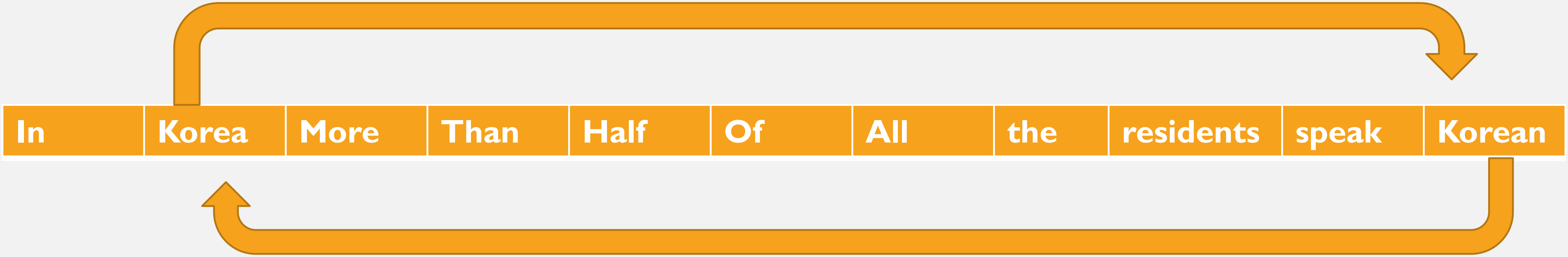
- Key Challenge: Long-Range Dependencies
- RNN: summarize all previous items in a hidden state
- LSTM: store information in a special memory cell

RNN: Summarize history

RNN										
In	Korea	More	Than	Half	Of	All	the	residents	speak	Korean
LSTM	Korea	Korea	Korea	Korea	Korea	Korea	Korea	Korea	Korea	

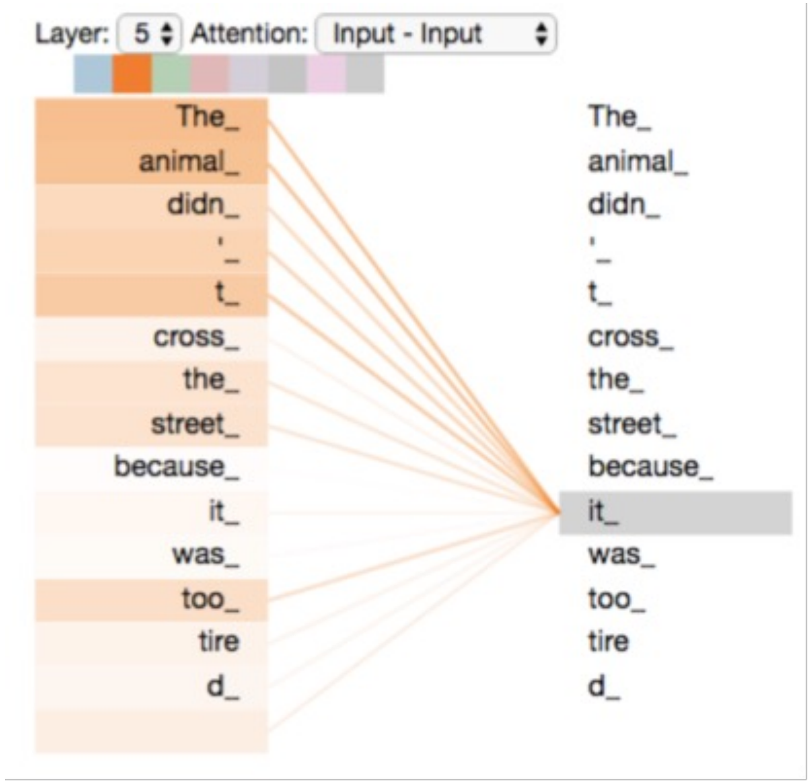
SELF-ATTENTION

- Self-Attention: Random Access Memory Model
- Every item has access to all other items in the sequence
  - Including future items



General formula: Encoding for word *i* is $encode(i) := \sum_j weight(i,j) \times value_j$

SELF-ATTENTION EXAMPLE



- The animal didn’t cross the street because it was tired
- Source: http://jalammar.github.io/illustrated-transformer/
- See also bertviz notebook

http://jalammar.github.io/illustrated-transformer/

COGNITIVE SCIENCE PERSPECTIVE

- A common model of human cognition posits a working memory
  - Similar to random-access memory in computers
- Working memory contains a finite set of "elements"
- Can relate each element to any other
  - May be a model of consciousness
- Random access = items are a set, not a sequence
  - Seems especially appropriate for natural language

SELF-ATTENTION: WORD ENCODING

SELF-ATTENTION: CONTEXTUALIZED ENCODING VIEW

High-level summary: given input sequence

1. Start with generic embeddings $word_1, \ldots, word_n$
2. Update each embedding given the word's *context*
   - Context = embeddings of other words
   - New encoding for word *i* is $encode'(i) := \sum_j weight(i,j) \times encode_j$
3. New encoding = new context
   - Repeat encoding update (*k* times)

ATTENTION WEIGHTS

- For seq-2-seq attention, we assigned weights to a pair (input position, output position).
- What are problems with this approach for self-attention?
  - Variable-length sequences →variable number of positions
  - Content is more important than position for relevance
- Example:
  - In Korea, most people speak Korean
  - Most people in Korea speak Korean
- How to solve these issues?

CONTENT-BASED ATTENTION WEIGHTS

- Let $[word_i]$, $[word_j]$ be the embedding of two words in the input sequence.
- How to measure their compatibility?
- Recall that dot product $\cdot$ between two word embeddings represents semantic similarity between two words

➢ $weight(word_i, word_j) = [word_i] \cdot [word_j]$ (?)

- Problem: dot product is symmetric but relevance is not
- Example: "In Korea, most residents speak Korean."
- "Korea" is very relevant to "Korean"
- "Korean" not so relevant to "Korea"

QUERY-KEY MODEL

- Linear transform of each embedding:

1. Produce a <u>query vector</u> $query_i := W^Q\ [word_i]$
2. a <u>key vector</u> $key_i := W^K\ [word_i]$

- Attention weight of word *j* for word *i:*
  $query_i \cdot key_j$
- Standardize and normalize to probabilities *weight(i,j)*
- [Visualization]

WORD ENCODING

- Also transform each embedding to $\underline{value}_i := W^V [word_i]$
- Encoding for word *i* is $encode(i) := \sum_j weight(i,j) \times value_j$

SELF-ATTENTION: SEQUENCE ENCODING AND DECODING

COMBINING WORD ENCODINGS

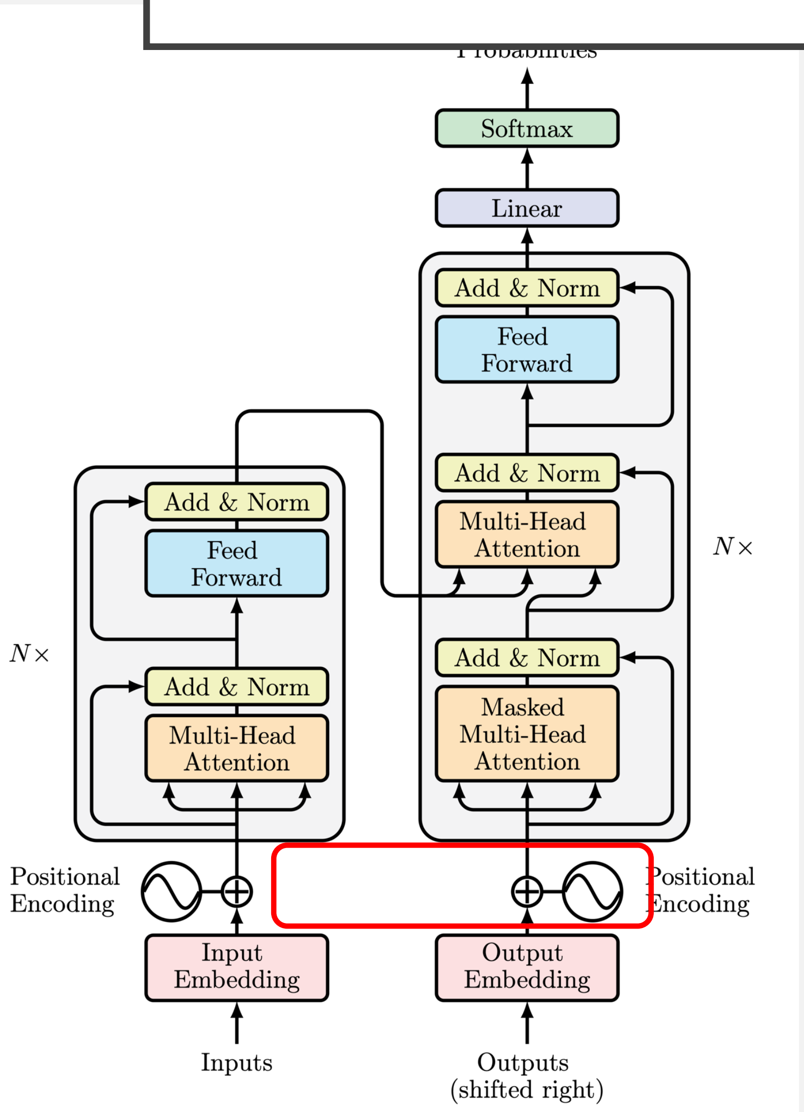
1. Each encoding vector *encode(i)* is given to the **same** feed-forward network to produce *z(i)*.
2. The encoding is repeated 6 times:
   1. Produce a new e'(i) given the current z(0), z(1),…,z(n)
   2. Input e'(i) to a feed-forward network to produce new z(i)

- [Visualization]()
- Final Output: key_i, $value_i$ for each input position

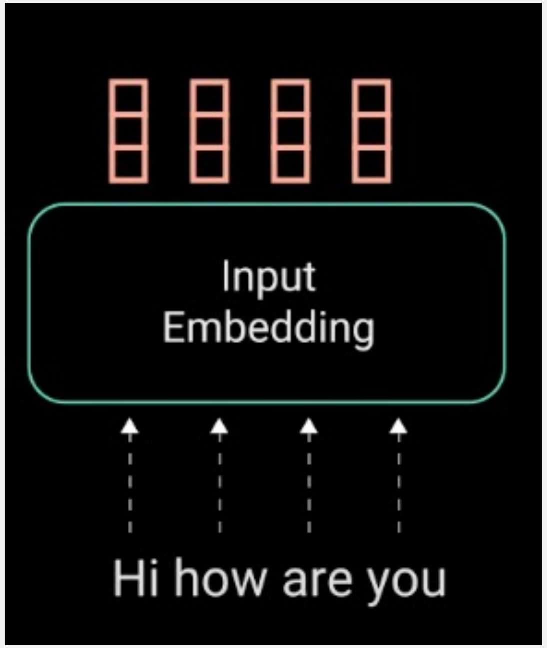
REFINEMENTS

- As if this were not complicated enough, you can also add
- Attention heads: multiple transformation matrices W^Q, W^K. W^V produce multiple encodings for each position
- Position encoding: as described the encoding loses all information about the position of the word.
  - Add a position encoding vector to each embedding *[word$_i$]*
- Normalize output values using layer normalization

POSITIONAL ENCODING



- Every position is mapped to a vector
- Bounded between -1,+1

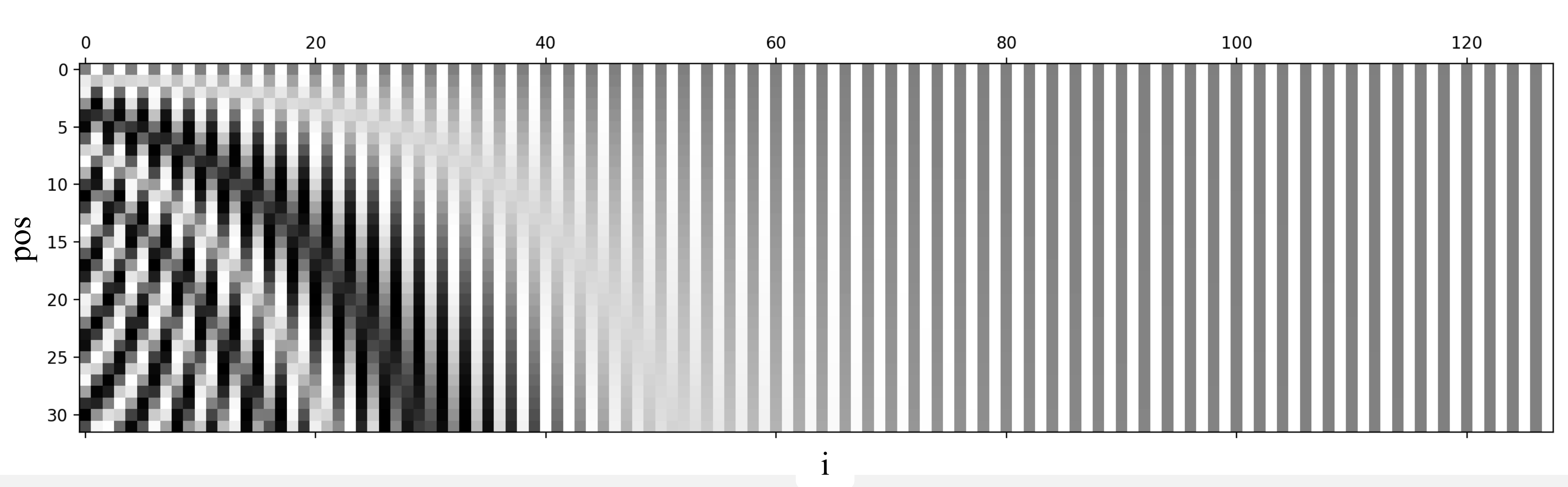


$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}})$$

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

Image Credit: [5, 7]

POSITIONAL ENCODING



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

Different waves for different positions

Image Credit: [8]

DECODING

- The decoder uses attention for input positions and self-attention from previous output positions
- Let *i* range over input positions, *j* over output positions.
- The *query(j)* vector is obtained from embedding [$\text{output\_word}_{(j-1)}$]
- Then $encode(j) := \sum_i weight(i,j) \times value_i + \sum_{j'<j} weight(j',j) \times value_{j'}$
  - The values and weights are computed using query(j), keys, and values as before.
- We obtain a final output vector *z(j)* as before

➢ A linear layer + softmax maps *z(j)* to a distribution over output words

CONCLUSION

- Self-attention is an alternative to RNN models (e.g. LSTM)
- Key limitation: requires $O(n^2)$ weights for n items
  - ➢ In practice, have to limit input size to a max constant
- Based on random access to all elements of the sequence
- Information from different sequence elements is combined using <u>attention weights</u>
- Transformer uses self-attention to encode input sequence, and to decode the output sequence