Contextualized Word Embeddings

CS114B Lab 12

Kenneth Lai

April 28, 2022

Contextualized Word Embeddings

CS114B Lab 12

Kenneth Lai

April 28, 2022

Source 1

Source 2

Contextualized Word Embeddings

CS114B Lab 12

Kenneth Lai

April 28, 2022

Source 1

Source 3

Distributed Representations of Words

- Representations of (contexts of) words as embeddings in some vector space
- Two approaches to distributed, distributional representations (Baroni et al. 2014):
  - Count-based
    - Count occurrences of words in contexts, optionally followed by some mathematical transformation (e.g. tf-idf, PPMI, SVD)
  - Prediction-based
    - Given some context vector(s) $\mathbf{c}$, predict some word $\mathbf{x}$ (or vice versa)
    - a.k.a. language modeling-based

Language Models

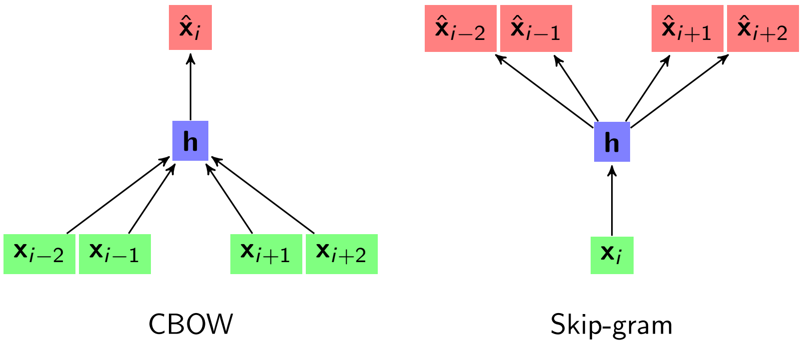
- Given some context vector(s) $\mathbf{c}$, predict some word $\mathbf{x}$ (or vice versa)
- Two approaches to language models:
  - Generative models
    - Model the joint probability distribution $P(\mathbf{x}, \mathbf{c})$
    - Examples: n-gram language models
      - Unigram: predict $P(\mathbf{x}_i)$
      - Bigram: predict $P(\mathbf{x}_i | \mathbf{x}_{i-1})$
      - Trigram: predict $P(\mathbf{x}_i | \mathbf{x}_{i-2}, \mathbf{x}_{i-1})$

Language Models

- Given some context vector(s) $\mathbf{c}$, predict some word $\mathbf{x}$ (or vice versa)
- Two approaches to language models:
  - Discriminative models
    - Predict the conditional probability $P(\mathbf{x}|\mathbf{c})$ (or $P(\mathbf{c}|\mathbf{x})$) directly
    - Examples: neural network language models
      - Feedforward: word2vec (Mikolov et al. 2013a, 2013b)
      - Recurrent:  (Peters et al. 2018)
      - Transformer:  (Devlin et al. 2019)

word2vec

- Based on a feedforward neural network language model



- Continuous bag of words (CBOW): use context to predict current word
- Skip-gram: use current word to predict context

word2vec

- Input layer: one-hot word vectors
- Hidden (projection) layer: identity activation function, no bias
  - Input $\rightarrow$ hidden $=$ table lookup (in weight matrix)
- Output layer: softmax activation function

word2vec

- Skip-gram model: for each word, word2vec learns two word embeddings
  - Target word vector $\mathbf{w}$ (row of $\mathbf{W}$, $=$ output of hidden layer)
  - "Context" word vector $\mathbf{c}$ (column of $\mathbf{C}$)

word2vec

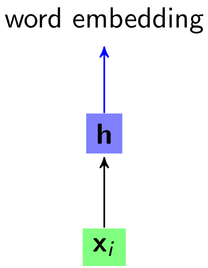
- Skip-gram model: for each word, word2vec learns two word embeddings
  - Target word vector $\mathbf{w}$ (row of $\mathbf{W}$, = output of hidden layer)
  - "Context" word vector $\mathbf{c}$ (column of $\mathbf{C}$)
    - Note that there is a single context word vector for each word, that is an abstraction over all contexts

word2vec

- Skip-gram model: for each word, word2vec learns two word embeddings
  - Target word vector $\mathbf{w}$ (row of $\mathbf{W}$, = output of hidden layer)
  - "Context" word vector $\mathbf{c}$ (column of $\mathbf{C}$)
    - Note that there is a single context word vector for each word, that is an abstraction over all contexts
- Common final word embeddings
  - Add $\mathbf{w} + \mathbf{c}$
  - Just $\mathbf{w}$ (throw away $\mathbf{c}$)

word2vec

- Skip-gram model: for each word, word2vec learns two word embeddings
  - Target word vector $\mathbf{w}$ (row of $\mathbf{W}$, = output of hidden layer)
  - "Context" word vector $\mathbf{c}$ (column of $\mathbf{C}$)
    - Note that there is a single context word vector for each word, that is an abstraction over all contexts
- Common final word embeddings
  - Add $\mathbf{w} + \mathbf{c}$
  - Just $\mathbf{w}$ (throw away $\mathbf{c}$)



word2vec

- Two issues with word2vec:

word2vec

- Two issues with word2vec:
  - One vector per word type

word2vec

- Two issues with word2vec:
    - One vector per word type
    - Limited (fixed-length) context

Polysemy

- One vector per word type

Polysemy

- One vector per word type
- But words have multiple senses

Polysemy

- One vector per word type
- But words have multiple senses
  - a mouse[1] controlling a computer system in 1968

Polysemy

- One vector per word type
- But words have multiple senses
  - a mouse[1] controlling a computer system in 1968
  - a quiet animal like a mouse[2]

Polysemy

- One vector per word type
- But words have multiple senses
  - a mouse^1 controlling a computer system in 1968
  - a quiet animal like a mouse^2
- Should mouse^1 and mouse^2 have the same word embedding?

Polysemy

- One vector per word type
- But words have multiple senses
  - a mouse^1 controlling a computer system in 1968
  - a quiet animal like a mouse^2
- Should mouse^1 and mouse^2 have the same word embedding?
  - Why not?

Polysemy

- One vector per word type
- But words have multiple senses
  - ... mouse^1 ... computer ...
  - ... animal ... mouse^2 ...
- Should mouse^1 and mouse^2 have the same word embedding?
  - Why not?
    - Syntagmatic association between mouse^1 and computer

Polysemy

- One vector per word type
- But words have multiple senses
  - ... mouse^1 ... computer ...
  - ... animal ... mouse^2 ...
- Should mouse^1 and mouse^2 have the same word embedding?
  - Why not?
    - Syntagmatic association between mouse^1 and computer
    - Syntagmatic association between mouse^2 and animal

Polysemy

- One vector per word type
- But words have multiple senses
  - ... mouse^1 ... computer ...
  - ... animal ... mouse^2 ...
- Should mouse^1 and mouse^2 have the same word embedding?
  - Why not?
    - Syntagmatic association between mouse^1 and computer
    - Syntagmatic association between mouse^2 and animal
    - Paradigmatic association between computer and animal!

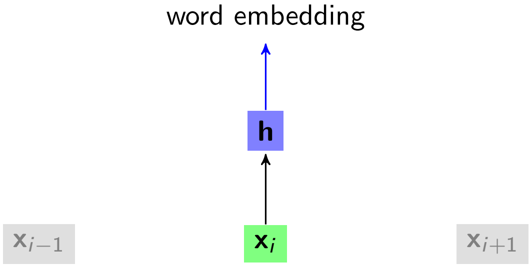
Polysemy

- One vector per word type
- But words have multiple senses
  - ... mouse^1 ... computer ...
  - ... animal ... mouse^2 ...
- Should mouse^1 and mouse^2 have the same word embedding?
  - Why not?
    - Syntagmatic association between mouse^1 and computer
    - Syntagmatic association between mouse^2 and animal
    - Paradigmatic association between computer and animal!
- How can we distinguish between mouse^1 and mouse^2?

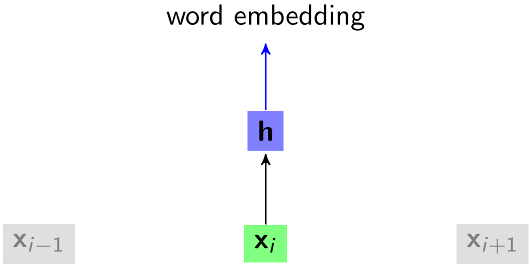
Polysemy

- One vector per word type
- But words have multiple senses
  - ... mouse^1 ... computer ...
  - ... animal ... mouse^2 ...
- Should mouse^1 and mouse^2 have the same word embedding?
  - Why not?
    - Syntagmatic association between mouse^1 and computer
    - Syntagmatic association between mouse^2 and animal
    - Paradigmatic association between computer and animal!
- How can we distinguish between mouse^1 and mouse^2?
  - Context!

Word Embeddings

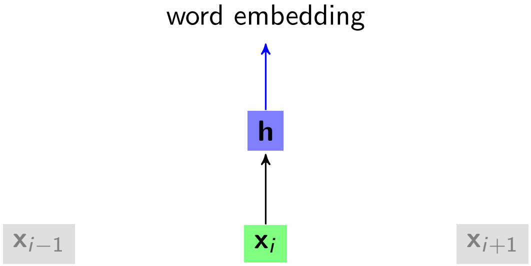


Word Embeddings



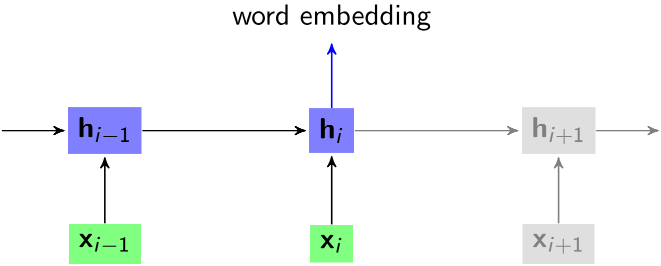
- **h** is an embedding of $\mathbf{x}_i$ only

Word Embeddings



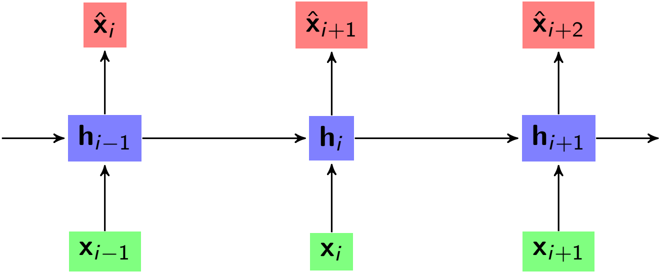
- $\mathbf{h}$ is an embedding of $\mathbf{x}_i$ only
  - How can we embed context information in $\mathbf{h}$?

Word Embeddings



- $\mathbf{h}$ is an embedding of $\mathbf{x}_i$ only
  - How can we embed context information in $\mathbf{h}$?

Recurrent Neural Networks



- Neural networks in which the output of a layer in one time step is input to a layer in the next time step

Recurrent Neural Networks

- RNNs allow for contextualized word embeddings

Recurrent Neural Networks

- RNNs allow for contextualized word embeddings
  - Multiple word senses

Recurrent Neural Networks

- RNNs allow for contextualized word embeddings
  - Multiple word senses
  - Arbitrary-length context

Recurrent Neural Networks

- RNNs allow for contextualized word embeddings
  - Multiple word senses
  - Arbitrary-length context
- Is this enough?

Context and Long-Distance Dependencies

- $\mathbf{h}_{i-1}$ encodes the context $\mathbf{x}_1, ..., \mathbf{x}_{i-1}$

Context and Long-Distance Dependencies

- $\mathbf{h}_{i-1}$ encodes the context $\mathbf{x}_1, \ldots, \mathbf{x}_{i-1}$
  - But mostly $\mathbf{x}_{i-1}$, less $\mathbf{x}_{i-2}$, even less $\mathbf{x}_{i-3}$, ..., very little $\mathbf{x}_1$

Context and Long-Distance Dependencies

- $\mathbf{h}_{i-1}$ encodes the context $\mathbf{x}_1, \ldots, \mathbf{x}_{i-1}$
  - But mostly $\mathbf{x}_{i-1}$, less $\mathbf{x}_{i-2}$, even less $\mathbf{x}_{i-3}$, ..., very little $\mathbf{x}_1$
- Context is local

Context and Long-Distance Dependencies

- Example: subject-verb agreement

Context and Long-Distance Dependencies

- Example: subject-verb agreement
- The flights the airline (was/were) cancelling (was/were) full.

Context and Long-Distance Dependencies

- Example: subject-verb agreement
- The flights the airline was cancelling (was/were) full.

Context and Long-Distance Dependencies

- Example: subject-verb agreement
- The flights the airline was cancelling (was/were) full.
  - The context for "was" is mostly "airline"

Context and Long-Distance Dependencies

- Example: subject-verb agreement
- The flights the airline was cancelling were full.
  - The context for "was" is mostly "airline"

Context and Long-Distance Dependencies

- Example: subject-verb agreement
- The flights the airline was cancelling were full.
  - The context for "was" is mostly "airline"
  - The context for "were" is mostly "cancelling", "was", "airline"

Context and Long-Distance Dependencies

- Example: subject-verb agreement
- The flights the airline was cancelling were full.
  - The context for “was” is mostly “airline”
  - The context for “were” is mostly “cancelling”, “was”, “airline”
    - Very little “flights”

Context and Long-Distance Dependencies

- Two approaches to handling long-distance dependencies:

Context and Long-Distance Dependencies

- Two approaches to handling long-distance dependencies:
  - Memory-based (e.g. long short-term)

Context and Long-Distance Dependencies

- Two approaches to handling long-distance dependencies:
  - Memory-based (e.g. long short-term)
    -  does this

Context and Long-Distance Dependencies

- Two approaches to handling long-distance dependencies:
  - Memory-based (e.g. long short-term)
    -  does this
  - Attention-based

Context and Long-Distance Dependencies

- Two approaches to handling long-distance dependencies:
  - Memory-based (e.g. long short-term)
    -  does this
  - Attention-based
    - At each time step, the model explicitly computes which other words to pay attention to

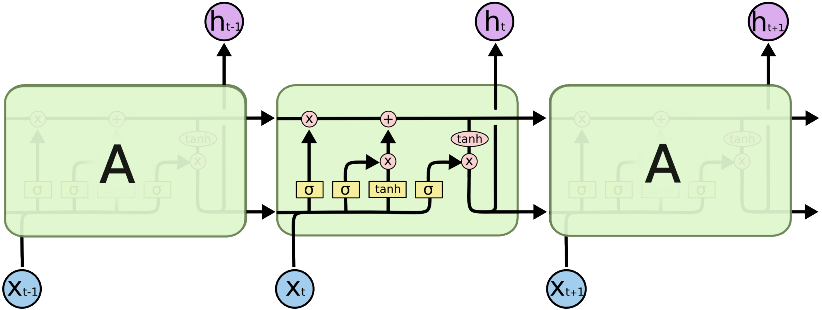
Context and Long-Distance Dependencies

- Two approaches to handling long-distance dependencies:
  - Memory-based (e.g. long short-term)
    -  does this
  - Attention-based
    - At each time step, the model explicitly computes which other words to pay attention to
    -  does this

- Embeddings from Language Models

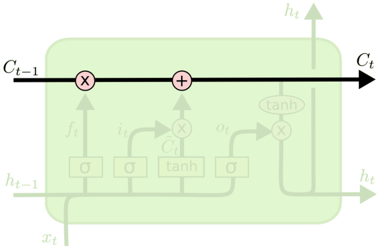
- Embeddings from Language Models
- Based on a bidirectional long short-term memory (LSTM) language model

Long Short-Term Memory



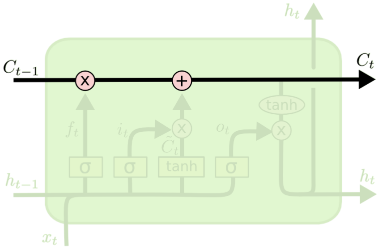
Source

Long Short-Term Memory



Source

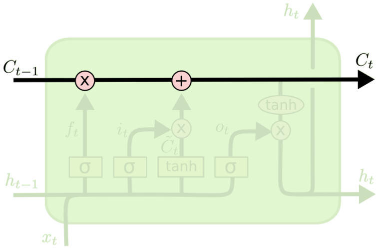
Long Short-Term Memory



Source

- Separate memory (cell) state

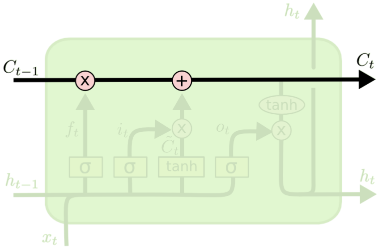
Long Short-Term Memory



Source

- Separate memory (cell) state
  - Reading from and writing to memory controlled by gates

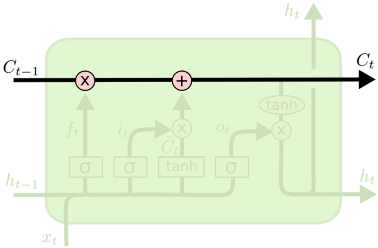
Long Short-Term Memory



Source

- Separate memory (cell) state
  - Reading from and writing to memory controlled by gates
    - Each gate contains one or two neural network layers

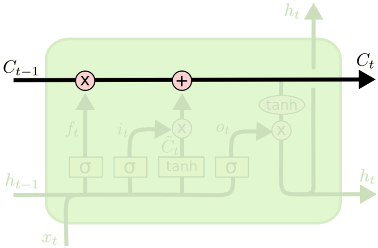
Long Short-Term Memory



Source

- Separate memory (cell) state
  - Reading from and writing to memory controlled by gates
    - Each gate contains one or two neural network layers
  - State persists across time

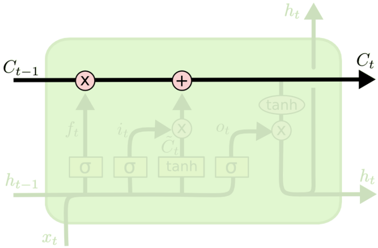
Long Short-Term Memory



Source

- Separate memory (cell) state
  - Reading from and writing to memory controlled by gates
    - Each gate contains one or two neural network layers
  - State persists across time
    - May remember information from long ago

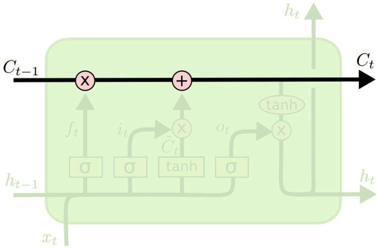
Long Short-Term Memory



Source

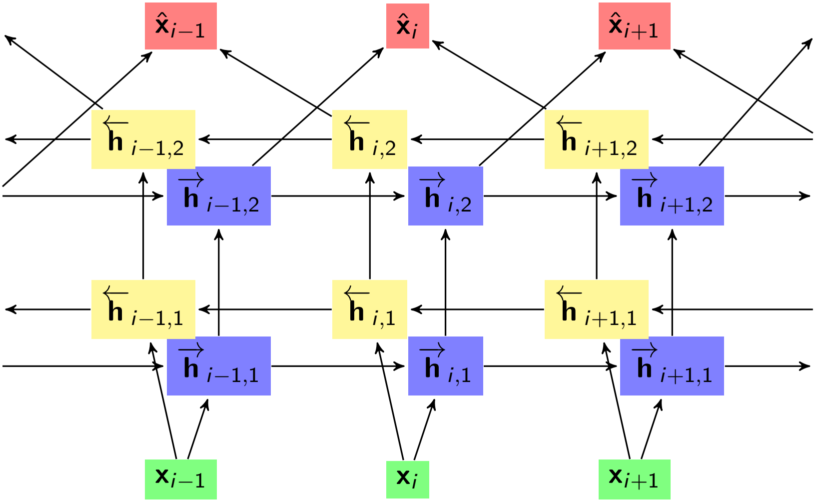
- Separate memory (cell) state
  - Reading from and writing to memory controlled by gates
    - Each gate contains one or two neural network layers
  - State persists across time
    - May remember information from long ago
    - Gradients for memory don't decay with time

Long Short-Term Memory



Source

- Separate memory (cell) state
  - Reading from and writing to memory controlled by gates
    - Each gate contains one or two neural network layers
  - State persists across time
    - May remember information from long ago
    - Gradients for memory don't decay with time
- See Christopher Olah's Understanding LSTM Networks for more details!



$\hat{\mathbf{x}}_{i-1}$
$\hat{\mathbf{x}}_{i}$
$\hat{\mathbf{x}}_{i+1}$
$\overleftarrow{\mathbf{h}}_{i-1,2}$
$\overleftarrow{\mathbf{h}}_{i,2}$
$\overleftarrow{\mathbf{h}}_{i+1,2}$
$\overrightarrow{\mathbf{h}}_{i-1,2}$
$\overrightarrow{\mathbf{h}}_{i,2}$
$\overrightarrow{\mathbf{h}}_{i+1,2}$
$\overleftarrow{\mathbf{h}}_{i-1,1}$
$\overleftarrow{\mathbf{h}}_{i,1}$
$\overleftarrow{\mathbf{h}}_{i+1,1}$
$\overrightarrow{\mathbf{h}}_{i-1,1}$
$\overrightarrow{\mathbf{h}}_{i,1}$
$\overrightarrow{\mathbf{h}}_{i+1,1}$
$\mathbf{x}_{i-1}$
$\mathbf{x}_{i}$
$\mathbf{x}_{i+1}$

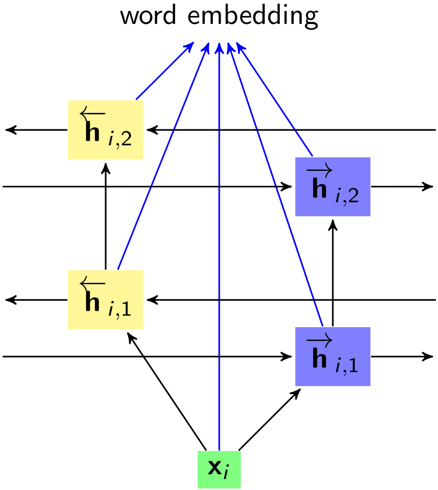
- Input layer: pre-trained word vectors (e.g. from word2vec)

- Input layer: pre-trained word vectors (e.g. from word2vec)
- 2 bidirectional LSTM layers

- Input layer: pre-trained word vectors (e.g. from word2vec)
- 2 bidirectional LSTM layers
- Output layer: softmax

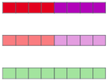
- Input layer: pre-trained word vectors (e.g. from word2vec)
- 2 bidirectional LSTM layers
- Output layer: softmax

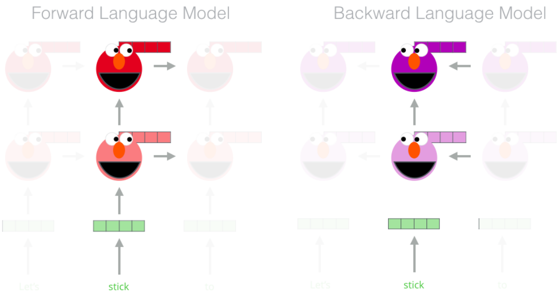
- Word embeddings: weighted sum of outputs of input and LSTM layers (task dependent)


word embedding
$\overleftarrow{\mathbf{h}}_{i,2}$
$\overrightarrow{\mathbf{h}}_{i,2}$
$\overleftarrow{\mathbf{h}}_{i,1}$
$\overrightarrow{\mathbf{h}}_{i,1}$
$\mathbf{x}_i$

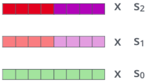
Embedding of “stick” in “Let’s stick to” - Step #2

1- Concatenate hidden layers





2- Multiply each vector by a weight based on the task



3- Sum the (now weighted) vectors

ELMo embedding of “stick” for this task in this context

Source

- Bidirectional Encoder Representations from Transformers

- Bidirectional Encoder Representations from Transformers
- Based on a transformer ("attention is all you need") model

Attention

1. Compute query, key, and value vectors for each input vector

Attention

1. Compute query, key, and value vectors for each input vector
   - Matrix multiplication

Attention

1. Compute query, key, and value vectors for each input vector
   - Matrix multiplication

- For each word i:

Attention

1. Compute query, key, and value vectors for each input vector
   - Matrix multiplication

- For each word i:
  2. Compute the dot product of query q_i with key k_j for each word j

Attention

1. Compute query, key, and value vectors for each input vector
   - Matrix multiplication

- For each word i:
  2. Compute the dot product of query q_i with key k_j for each word j
  3. Scale the dot product (e.g. Vaswani et al. (2017) divide by 8)
     - Leads to more stable gradients

Attention

1. Compute query, key, and value vectors for each input vector
   - Matrix multiplication

- For each word i:

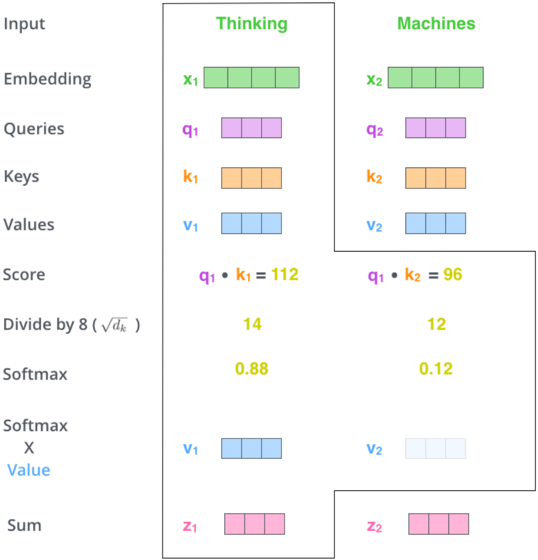
2. Compute the dot product of query q_i with key k_j for each word j
3. Scale the dot product (e.g. Vaswani et al. (2017) divide by 8)
   - Leads to more stable gradients
4. Softmax

Attention

1. Compute query, key, and value vectors for each input vector
   - Matrix multiplication

- For each word i:
  2. Compute the dot product of query q_i with key k_j for each word j
  3. Scale the dot product (e.g. Vaswani et al. (2017) divide by 8)
     - Leads to more stable gradients
  4. Softmax
  5. Compute the weighted sum of values v_j for each word j
     - Weights = softmax output from previous step

Attention



Source

Attention

- Output: weighted sum of value vectors (modulo some more advanced topics)

Attention

- Output: weighted sum of value vectors (modulo some more advanced topics)
  - Multi-head attention
  - Positional encodings
  - Residual connections
  - Layer normalization

Attention

- Output: weighted sum of value vectors (modulo some more advanced topics)
  - Multi-head attention
  - Positional encodings
  - Residual connections
  - Layer normalization
- See Jay Alammar's The Illustrated Transformer for more details!

Transformers

- "Attention Is All You Need" (Vaswani et al. 2017)

Transformers

- "Attention Is All You Need" (Vaswani et al. 2017)
- No recurrence, relies entirely on attention (and feedforward layers) to capture global dependencies

Transformers

- "Attention Is All You Need" (Vaswani et al. 2017)
- No recurrence, relies entirely on attention (and feedforward layers) to capture global dependencies
  - Recurrent neural networks are inherently sequential, processing one word at a time

Transformers

- "Attention Is All You Need" (Vaswani et al. 2017)
- No recurrence, relies entirely on attention (and feedforward layers) to capture global dependencies
  - Recurrent neural networks are inherently sequential, processing one word at a time
  - Transformers are more parallel, looking at the entire sequence at once

Transformers

- "Attention Is All You Need" (Vaswani et al. 2017)
- No recurrence, relies entirely on attention (and feedforward layers) to capture global dependencies
  - Recurrent neural networks are inherently sequential, processing one word at a time
  - Transformers are more parallel, looking at the entire sequence at once
    - More efficient, especially on GPUs

Transformers

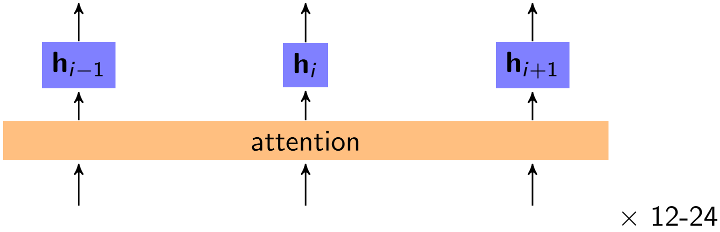
- "Attention Is All You Need" (Vaswani et al. 2017)
- No recurrence, relies entirely on attention (and feedforward layers) to capture global dependencies
  - Recurrent neural networks are inherently sequential, processing one word at a time
  - Transformers are more parallel, looking at the entire sequence at once
    - More efficient, especially on GPUs
    - Also scores better on many NLP tasks

- Input layer: pre-trained word vectors (e.g. from word2vec)

- Input layer: pre-trained word vectors (e.g. from word2vec)
- 12-24 encoder layers

- Input layer: pre-trained word vectors (e.g. from word2vec)
- 12-24 encoder layers
  - Encoder layer = (shared) attention layer + (individual) feedforward layers

- Input layer: pre-trained word vectors (e.g. from word2vec)
- 12-24 encoder layers
  - Encoder layer = (shared) attention layer + (individual) feedforward layers



- Output layer: 2 pre-training tasks

- Output layer: 2 pre-training tasks
    - Masked LM (Cloze)

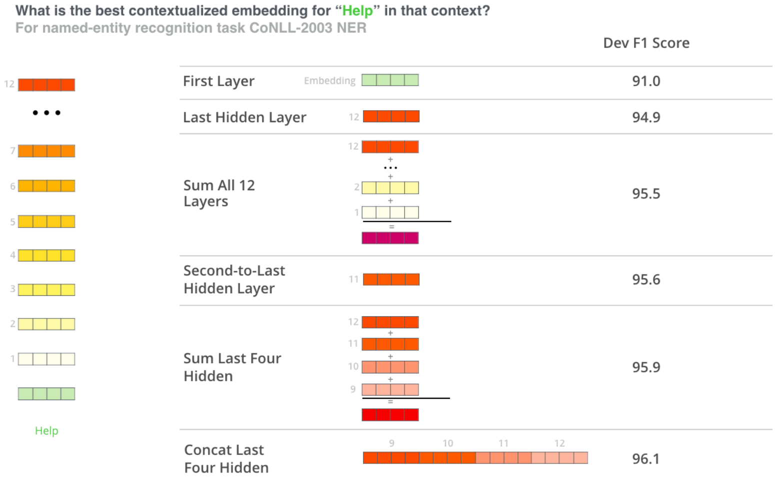
- Output layer: 2 pre-training tasks
    - Masked LM (Cloze)
        - Mask 15% of input tokens at random, predict masked words

- Output layer: 2 pre-training tasks
  - Masked LM (Cloze)
    - Mask 15% of input tokens at random, predict masked words
  - NSP (Next Sentence Prediction)

- Output layer: 2 pre-training tasks
  - Masked LM (Cloze)
    - Mask 15% of input tokens at random, predict masked words
  - NSP (Next Sentence Prediction)
    - Given sentences A and B, does B follow A?

- Word embeddings: combinations of outputs of encoder layers

- Word embeddings: combinations of outputs of encoder layers



Source

References

- Baroni, Marco, Georgiana Dinu, and Germán Kruszewski. "Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors." Proceedings of ACL. 2014.
- Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." Proceedings of NAACL-HLT. 2019.
- Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean. "Efficient estimation of word representations in vector space." Proceedings of ICLR. 2013.
- Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. "Distributed representations of words and phrases and their compositionality." Proceedings of NeurIPS. 2013.
- Peters, Matthew E., Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. "Deep Contextualized Word Representations." Proceedings of NAACL-HLT. 2018.
- Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. "Attention is all you need." Proceedings of NeurIPS. 2017.