

Contextualized Word Embeddings (and Large Language Models in general)

CS114B Lab 12

Kenneth Lai

April 21, 2023

Contextualized Word Embeddings (and Large Language Models in general)

CS114B Lab 12

Kenneth Lai

April 21, 2023



Source 1



Source 2

Contextualized Word Embeddings (and Large Language Models in general)

CS114B Lab 12

Kenneth Lai

April 21, 2023



Source 1



Source 3

Contextualized Word Embeddings (and Large Language Models in general)

CS114B Lab 12

Kenneth Lai

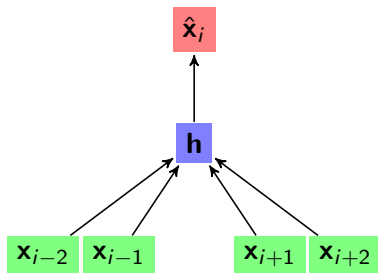
April 21, 2023



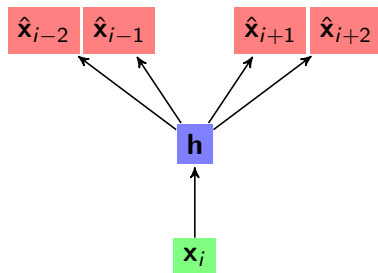
Now with GPT and friends!

word2vec

- Based on a feedforward neural network language model



CBOW



Skip-gram

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

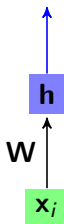
word2vec

- ▶ Training data: Google News (6 billion words)
- ▶ 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})
- ▶ Common final word embeddings
 - ▶ Add $\mathbf{w} + \mathbf{c}$
 - ▶ Just \mathbf{w} (throw away \mathbf{c})

word2vec

- ▶ Training data: Google News (6 billion words)
- ▶ 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})
- ▶ Common final word embeddings
 - ▶ Add $\mathbf{w} + \mathbf{c}$
 - ▶ Just \mathbf{w} (throw away \mathbf{c})

word embedding



word2vec

- ▶ Two issues with word2vec:
 - ▶ One vector per word type
 - ▶ Limited (fixed-length) context
 - ▶ e.g., ± 2 words, etc.

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?

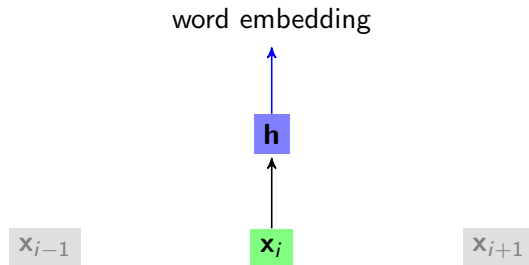
Polysemy

- ▶ One vector per word type
- ▶ But words have multiple senses
 - ▶ ... mouse¹ ... computer ...
 - ▶ ... animal ... mouse² ...
- ▶ Should mouse¹ and mouse² have the same word embedding?
 - ▶ Embeddings of computer and animal wind up closer than they “should” be

Polysemy

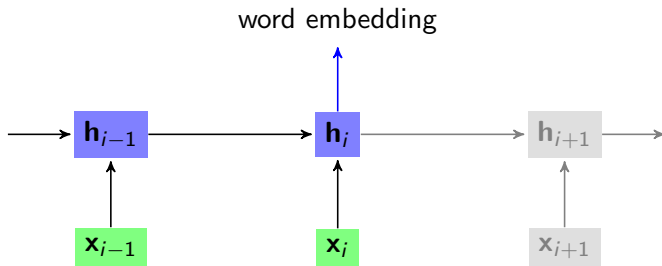
- ▶ One vector per word type
- ▶ But words have multiple senses
 - ▶ ... mouse¹ ... computer ...
 - ▶ ... animal ... mouse² ...
- ▶ Should mouse¹ and mouse² have the same word embedding?
 - ▶ Embeddings of computer and animal wind up closer than they “should” be
- ▶ How can we distinguish between mouse¹ and mouse²?
 - ▶ Context!

Word Embeddings



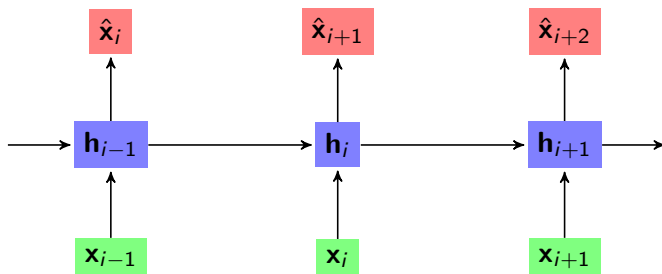
- ▶ h is an embedding of x_i only
 - ▶ How can we embed context information in h ?

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
 - ▶ Here, time step = word

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 encodes the context $\mathbf{x}_1, \dots, \mathbf{x}_i$
 - ▶ But mostly \mathbf{x}_i , less \mathbf{x}_{i-1} , even less \mathbf{x}_{i-2} , ..., very little \mathbf{x}_1
- ▶ Context is **local**

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.
 - ▶ 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**”
 - ▶ Very little “**flights**”

Context and Long-Distance Dependencies

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



- ▶ Elmo does this

- ▶ Attention-based

- ▶ At each time step, the model explicitly computes which other words to pay attention to

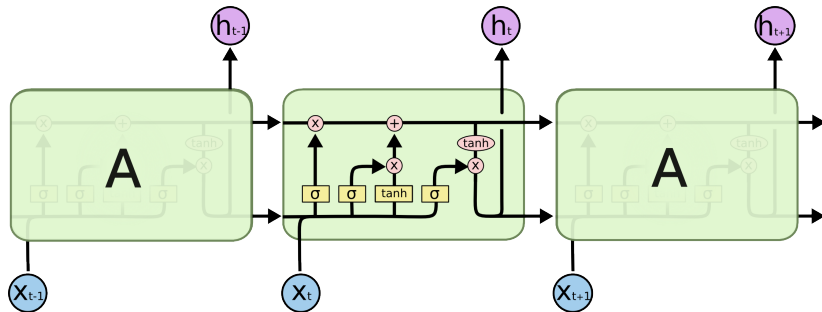


- ▶ Bert, GPT do this



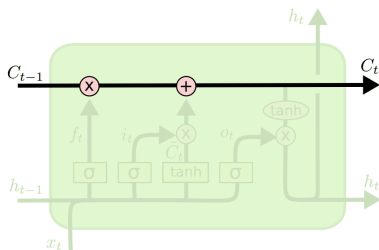
- ▶ Embeddings from Language Models
- ▶ Based on a bidirectional long short-term memory (LSTM) language model

Long Short-Term Memory



Source

Long Short-Term Memory

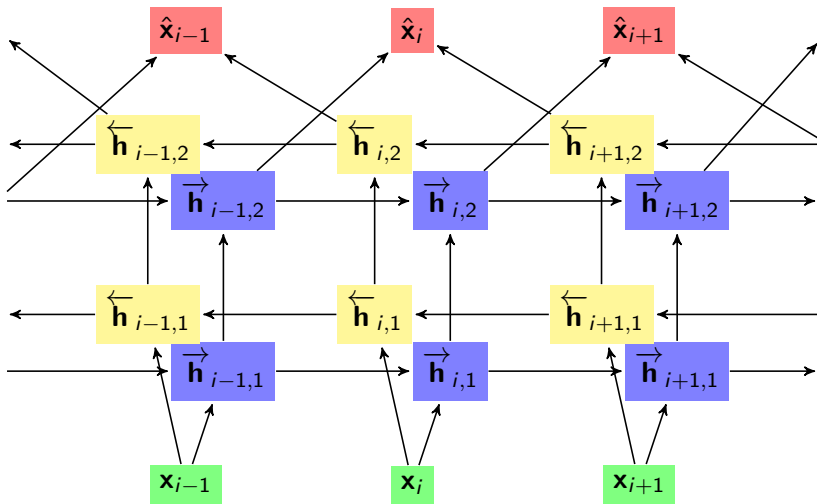


Source

- ▶ 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
- ▶ See Christopher Olah's **Understanding LSTM Networks** for more details!

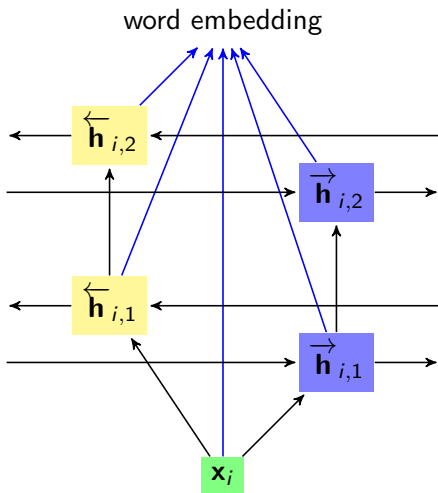


- ▶ Input layer: static (context-insensitive) word vectors
 - ▶ Character n-gram convolutions
- ▶ 2 bidirectional LSTM layers
- ▶ Output layer: softmax





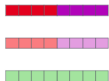
- ▶ Training data: Billion Word Benchmark
- ▶ Word embeddings: weighted sum of outputs of input and LSTM layers (task dependent)





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

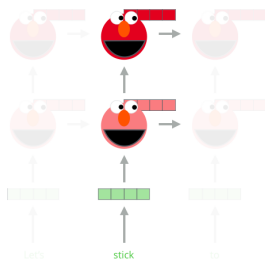


3- Sum the (now weighted) vectors

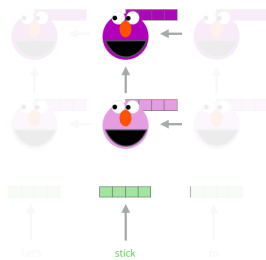


ELMo embedding of “stick” for this task in this context

Forward Language Model



Backward Language Model



Source



- ▶ Bidirectional Encoder Representations from Transformers
- ▶ Based on a transformer (“attention is all you need”) model
 - ▶ See Jay Alammar’s [The Illustrated Transformer](#) for more details!

Attention

Input

Embedding

Queries

Keys

Values

Score

Divide by $8 (\sqrt{d_k})$

Softmax


Softmax

X

Value

Sum

Thinking

x_1 

q_1 

k_1 

v_1 

$q_1 \cdot k_1 = 112$


14

0.88

v_1 

z_1 

Machines

x_2 

q_2 


k_2 

v_2 

$q_2 \cdot k_2 = 96$

12

0.12

v_2 

z_2 

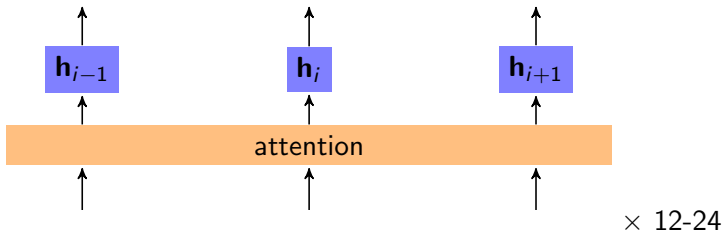
Source

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: static WordPiece vectors
 - ▶ WordPieces: subwords generated similarly to byte-pair encoding
- ▶ 12-24 encoder layers
 - ▶ Encoder layer = (shared) attention layer + (individual) feedforward layers



Applying Language Models to Downstream Tasks

- ▶ Three general approaches:

1. Feature-based

- ▶ Extract word embeddings from the language model, and use them as input features to a separate task-specific model



- ▶ word2vec, do this

Applying Language Models to Downstream Tasks

- ▶ Three general approaches:

- 1. Feature-based

- ▶ Extract word embeddings from the language model, and use them as input features to a separate task-specific model



- ▶ word2vec, do this

- 2. Fine-tuning

- ▶ Replace the output layer of the language model with a different head, and continue training, updating some or all parameters



- ▶ , GPT-1 do this



- ▶ 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 ?
- ▶ Training data:
 - ▶ BookCorpus (800 million words)
 - ▶ English Wikipedia (2.5 billion words)



► Word embeddings: combinations of outputs of encoder layers

What is the best contextualized embedding for “Help” in that context?

For named-entity recognition task CoNLL-2003 NER

12

...

7

6

5

4

3

2

1

Help

		Dev F1 Score
First Layer	Embedding	91.0
Last Hidden Layer	12	94.9
Sum All 12 Layers	 12 + 11 + 10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 + 4	95.5
Second-to-Last Hidden Layer	11	95.6
Sum Last Four Hidden	 12 + 11 + 10 + 9 + 4	95.9
Concat Last Four Hidden	 9 10 11 12 	96.1

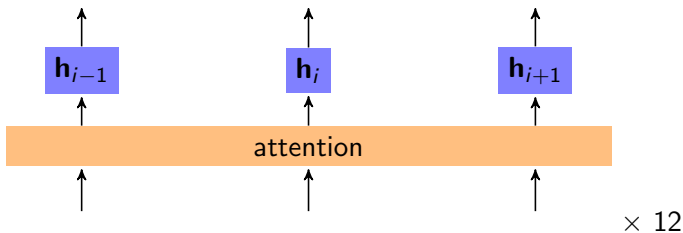
Source



- ▶ Generative Pre-Training, or
Generative Pre-trained Transformer
- ▶ Also based on a transformer model



- ▶ Input layer: BPE vocabulary
- ▶ 12 decoder layers
 - ▶ Decoder layer = (shared) attention layer + (individual) feedforward layers



- ▶ Decoders are unidirectional: mask out attention to future time steps



- ▶ Output layer:
 - ▶ Pre-training: standard language modeling
 - ▶ Fine-tuning: task-specific head, with language modeling as auxiliary objective
- ▶ Training data: BookCorpus (800 million words)



- ▶ GPT-2
 - ▶ More parameters: 117 million → up to 1.5 billion
 - ▶ More data: WebText (8 million documents, <19 billion tokens)
 - ▶ In-context learning

Applying Language Models to Downstream Tasks

- ▶ Three general approaches:

- 2. Fine-tuning

- ▶ Replace the output layer of the language model with a different head, and continue training, updating some or all parameters



- ▶ , GPT-1 do this

Applying Language Models to Downstream Tasks

- ▶ Three general approaches:

- 2. Fine-tuning

- ▶ Replace the output layer of the language model with a different head, and continue training, updating some or all parameters



- ▶ , GPT-1 do this

- 3. In-context (few-shot/one-shot/zero-shot) learning

- ▶ At inference (test) time, the model is given a description of the task in its context window, a few/one/zero demonstrations of the task, and asked to complete the text
 - ▶ Everything is language modeling!
 - ▶ GPT-2 and up do this

Applying Language Models to Downstream Tasks

The three settings we explore for in-context learning

Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.

```
1 Translate English to French: ← task description
2 cheese => ..... ← prompt
```

One-shot

In addition to the task description, the model sees a single example of the task. No gradient updates are performed.

```
1 Translate English to French: ← task description
2 sea otter => loutre de mer ← example
3 cheese => ..... ← prompt
```

Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

```
1 Translate English to French: ← task description
2 sea otter => loutre de mer ← examples
3 peppermint => menthe poivrée ←
4 plush giraffe => girafe peluche ←
5 cheese => ..... ← prompt
```

Traditional fine-tuning (not used for GPT-3)

Fine-tuning

The model is trained via repeated gradient updates using a large corpus of example tasks.





▶ GPT-3

- ▶ More parameters: 1.5 billion → up to 175 billion
- ▶ More data:
 - ▶ Common Crawl (filtered) (410 billion tokens)
 - ▶ WebText2 (19 billion tokens)
 - ▶ Books1 (12 billion tokens)
 - ▶ Books2 (55 billion tokens)
 - ▶ English Wikipedia (3 billion tokens)



▶ GPT-3

- ▶ More parameters: 1.5 billion → up to 175 billion
- ▶ More data:
 - ▶ Common Crawl (filtered) (410 billion tokens)
 - ▶ WebText2 (19 billion tokens)
 - ▶ Books1 (12 billion tokens)
 - ▶ Books2 (55 billion tokens)
 - ▶ English Wikipedia (3 billion tokens)

▶ GPT-3.5 (i.e., InstructGPT, ChatGPT)

- ▶ Reinforcement learning from human feedback (RLHF)

Reinforcement Learning from Human Feedback

- ▶ Reinforcement learning
 - ▶ Environment modeled as a Markov decision process
 - ▶ Set of states
 - ▶ Set of actions
 - ▶ Transition probabilities of going from state s from state s' by executing action a
 - ▶ Rewards received after going from state s from state s' by executing action a
 - ▶ Goal: learn an optimal policy
 - ▶ What action(s) to execute at each state

Reinforcement Learning from Human Feedback

- ▶ Reinforcement learning
 - ▶ Environment modeled as a Markov decision process
 - ▶ Set of states
 - ▶ Set of actions
 - ▶ Transition probabilities of going from state s from state s' by executing action a
 - ▶ Rewards received after going from state s from state s' by executing action a
 - ▶ Goal: learn an optimal policy
 - ▶ What action(s) to execute at each state
- ▶ For us:
 - ▶ State \approx context (prompt)
 - ▶ Action \approx model output
 - ▶ Policy \approx the language model itself

Reinforcement Learning from Human Feedback

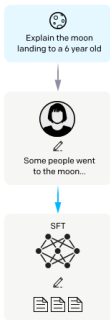
Step 1

Collect demonstration data, and train a supervised policy.

A prompt is sampled from our prompt dataset.

A labeler demonstrates the desired output behavior.

This data is used to fine-tune GPT-3 with supervised learning.



Step 2

Collect comparison data, and train a reward model.

A prompt and several model outputs are sampled.

A labeler ranks the outputs from best to worst.

This data is used to train our reward model.



Step 3

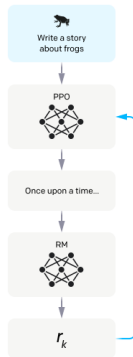
Optimize a policy against the reward model using reinforcement learning.

A new prompt is sampled from the dataset.

The policy generates an output.

The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.



Source



- ▶ GPT-4
 - ▶ More parameters (???)
 - ▶ More data (???)



► Multimodal (text and image) input

User What is funny about this image? Describe it panel by panel.



Source: [hmmmm \(Reddit\)](#)

GPT-4 The image shows a package for a "Lightning Cable" adapter with three panels.

Panel 1: A smartphone with a VGA connector (a large, blue, 15-pin connector typically used for computer monitors) plugged into its charging port.

Panel 2: The package for the "Lightning Cable" adapter with a picture of a VGA connector on it.

Panel 3: A close-up of the VGA connector with a small Lightning connector (used for charging iPhones and other Apple devices) at the end.

The humor in this image comes from the absurdity of plugging a large, outdated VGA connector into a small, modern smartphone charging port.