

# Encoder-Decoder Attention

---

MICHAEL WOLLOWSKI

SUMMARY OF CHAPTER 9: RNNs AND LSTMS

FROM: SPEECH AND LANGUAGE PROCESSING.

BY JURAFSKY AND MARTIN. [HTTPS://WEB.STANFORD.EDU/~JURAFSKY/SLP3/](https://web.stanford.edu/~jurafsky/slp3/)

1

## Limitations of RNNs

---

RNNs are good at processing sequences such as languages.

However, they have a drawback.

Consider the statement: “The flights the airline was cancelling were full.”

- What does “was” refer to?
  - “airline” i.e. the prior word
- What does “were” refer to?
  - “flights” i.e. a word much earlier in the sentence

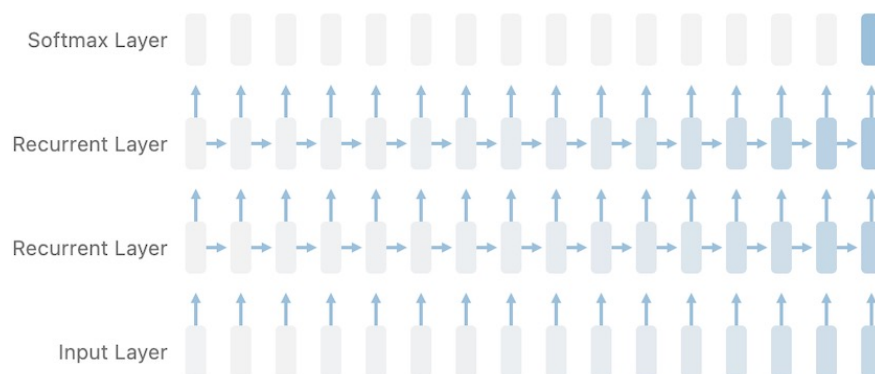
In NLP, one has to “remember” data:

- from the recent past as well as
- potentially from the more distant past.

2

## Limitations of RNNs

Vanishing gradients.



Source: <https://distill.pub/2019/memorization-in-rnns/>

3

## Long-Short Term Memory (LSTM) Nets

To address the limitations of RNNs, more complex units were developed, called LSTMs.

Those units are designed to explicitly manage a kind of memory

They are RNNs with an additional “context” path

In other words, they have two means of maintaining/remembering information:

- the recurrent state information
- context data, maintained by the network.

4

## Long-Short Term Memory (LSTM) Nets

In Long short-term memory (LSTM) networks, the units are designed to:

- remove information from the context that is no longer needed, and
- add information to the context that is likely to be relevant for later processing.

The units use *gates* to control the flow of information into and out of the units.

5

## LSTM Units in Detail

Here is an overview of an LSTM unit

Notice that there are

- network layers
- flow of data
- Vector multiplication (element-wise) and addition

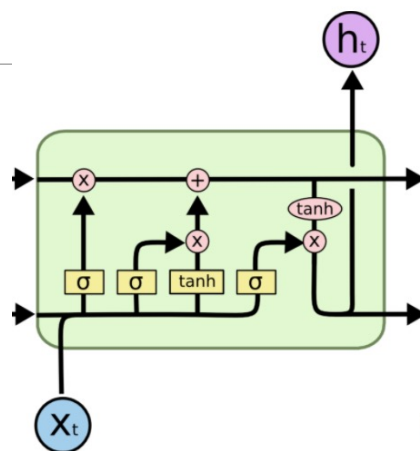
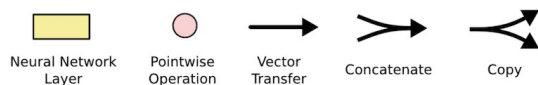


Image source: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

6

## Common RNN NLP Architectures

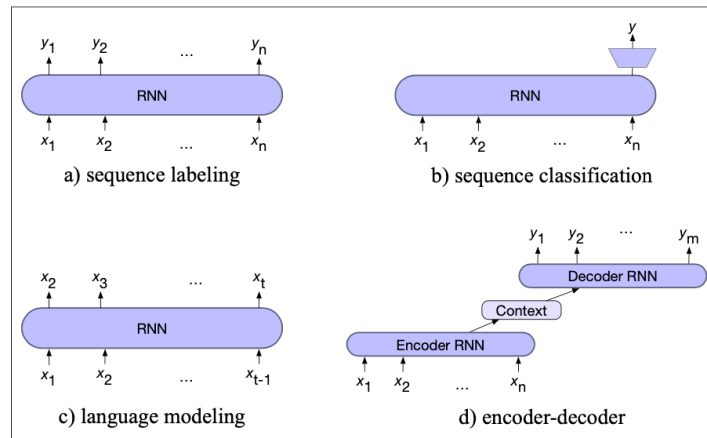


**Figure 9.15** Four architectures for NLP tasks. In sequence labeling (POS or named entity tagging) we map each input token  $x_i$  to an output token  $y_i$ . In sequence classification we map the entire input sequence to a single class. In language modeling we output the next token conditioned on previous tokens. In the encoder model we have two separate RNN models, one of which maps from an input sequence  $x$  to an intermediate representation we call the **context**, and a second of which maps from the context to an output sequence  $y$ .

Image source: Speech and Language Processing. Daniel Jurafsky & James H. Martin. Draft of February 3, 2024.

7

## Encoder-Decoder

- Also called sequence-to-sequence network
- Used when an input sequence is to be translated to an output sequence.
- Handy, when they are of a different lengths.
- Example: machine translation

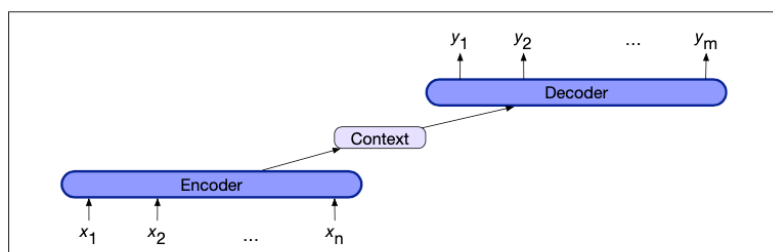


**Figure 9.16** The encoder-decoder architecture. The context is a function of the hidden representations of the input, and may be used by the decoder in a variety of ways.

Image source: Speech and Language Processing. Daniel Jurafsky & James H. Martin. Draft of February 3, 2024.

8

# Encoder-Decoder

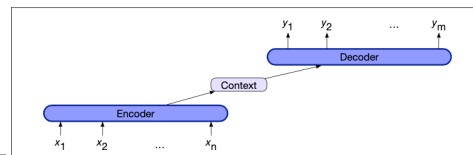


**Figure 9.16** The encoder-decoder architecture. The context is a function of the hidden representations of the input, and may be used by the decoder in a variety of ways.

The encoder network takes an input sequence and creates a contextualized representation of it, called the *context*.

At this point, you can think of the context vector produced by an LSTM network.

Encoder-decoder networks consist of three conceptual components:

1. An encoder that processes an input sequence,  $x_{1:n}$ , and generates a corresponding sequence of contextualized representations.
2. A context vector,  $c$ , which captures the essence of the input.
3. A decoder, which takes  $c$  as input generates an arbitrary length output sequence.

Image source: Speech and Language Processing. Daniel Jurafsky & James H. Martin. Draft of February 3, 2024.

9

# Encoder-Decoder

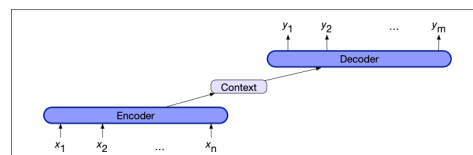


**Figure 9.16** The encoder-decoder architecture. The context is a function of the hidden representations of the input, and may be used by the decoder in a variety of ways.

Challenge: The context vector must represent absolutely everything about the source text to facilitate the task at hand.

It is somewhat as a bottleneck:

- It must contain information from the beginning of the sentence,
- As well as from the more recent portion of the sentence.

In other words, how much information can one pack into this vector?

Image source: Speech and Language Processing. Daniel Jurafsky & James H. Martin. Draft of February 3, 2024.

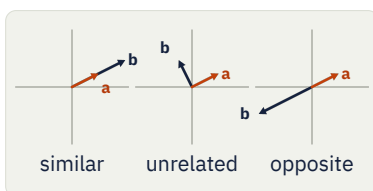
10

## Dot Product to Calculate Similarity

### MULTIPLY AND ADD

$$a \cdot b = a_1b_1 + a_2b_2 + \dots + a_nb_n$$

Pair up the entries of the two vectors, multiply each pair, then add the results.



### READ IT AS SIMILARITY

- Large and positive: the vectors are similar
- Zero: they are unrelated
- Negative: they are opposites

Worked example:  $a = (2, 1)$  compared with three choices of  $b$

| Case      | $b$        | Multiply and add              | Score |
|-----------|------------|-------------------------------|-------|
| Similar   | $(4, 2)$   | $2 \cdot 4 + 1 \cdot 2$       | 10    |
| Unrelated | $(-1, 2)$  | $2 \cdot (-1) + 1 \cdot 2$    | 0     |
| Opposite  | $(-4, -2)$ | $2 \cdot (-4) + 1 \cdot (-2)$ | -10   |

15

## Dot Product to Calculate Similarity

Decoder state  $h = (1, 1)$ ; each input word has a state  $s$ . Toy numbers.

| Word  | State $s$ | Score $h \cdot s$           | $\exp(\text{score})$ | Weight |
|-------|-----------|-----------------------------|----------------------|--------|
| The   | $(0, 1)$  | $1 \cdot 0 + 1 \cdot 1 = 1$ | 2.718                | 0.09   |
| cat   | $(2, 1)$  | $1 \cdot 2 + 1 \cdot 1 = 3$ | 20.086               | 0.67   |
| sat   | $(1, 1)$  | $1 \cdot 1 + 1 \cdot 1 = 2$ | 7.389                | 0.24   |
| Total |           |                             | 30.193               | 1.00   |

$\exp(\text{score})$  means  $e$  raised to the score ( $e \approx 2.718$ ): it makes every score positive and boosts the larger ones. Each weight is  $\exp(\text{score}) \div \text{the column total}$ , 30.193, so the weights add up to 1.

- “cat” gets 67% of the attention because its state scores highest against  $h$ .
- “sat” scores second, and
- “The” barely registers.

16

## Why attention? The RNN bottleneck

### WITHOUT ATTENTION

The cat sat → context → decoder

One fixed-size vector must hold the whole sentence, and the decoder sees nothing else.

### WITH ATTENTION

The cat sat → blend → decoder  
 9% 67% 24%

At every step, the decoder (state  $h$ ) asks which input words matter right now: it scores each state  $s$  with a dot product.

### 1 · SCORE EACH WORD

score =  $h \cdot s$

### 2 · SCORES TO WEIGHTS

weights = softmax(scores)

### 3 · BLEND THE STATES

context =  $\sum \text{weight} \times s$

### STEP 3 WORKED OUT: THE CONTEXT VECTOR

$0.09 \cdot (0, 1) + 0.67 \cdot (2, 1) + 0.24 \cdot (1, 1) = (1.58, 1.00)$

17

## Attention

The attention mechanism is a solution to the bottleneck problem.

It collects information from *all* the hidden states of the encoder, not just the last hidden state.

In the attention mechanism, the context vector  $c$  is a function of the hidden states of the encoder:  
 $c = f(h^e_1 \dots h^e_n)$ .

We create a single fixed-length vector  $c$  by taking a weighted sum of all the encoder hidden states.

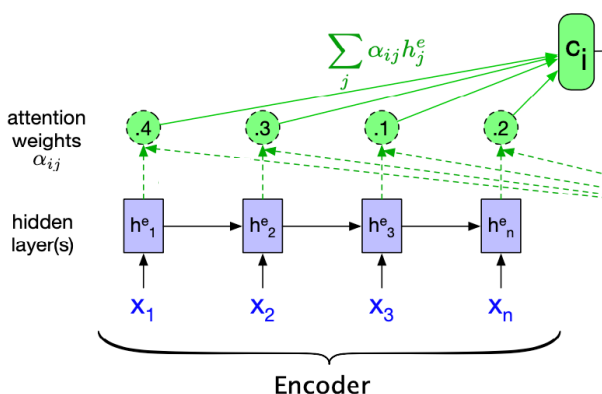
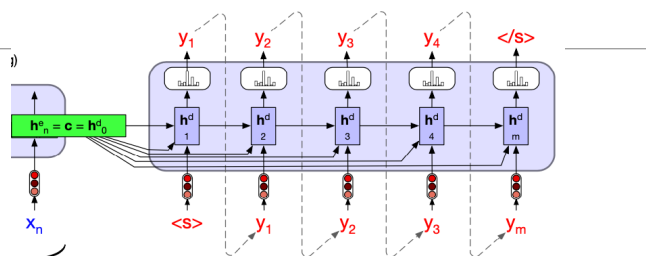


Image source: Speech and Language Processing. Daniel Jurafsky & James H. Martin. Draft of February 3, 2024.

19

# Attention



This context is made available during decoding, along with the prior hidden state and the previous output generated by the decoder.

Image source: Speech and Language Processing. Daniel Jurafsky & James H. Martin. Draft of February 3, 2024.

20

# Attention

- For more fine-grained context the vector can be generated anew with each decoding step.
- The decoder would have its own weights

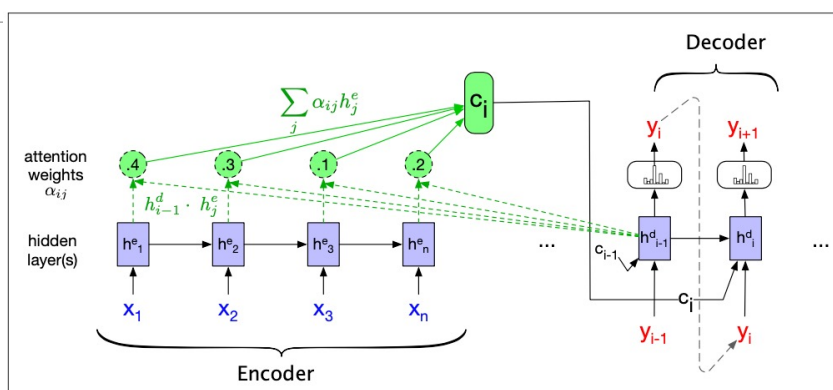


**Figure 9.22** A sketch of the encoder-decoder network with attention, focusing on the computation of  $c_i$ . The context value  $c_i$  is one of the inputs to the computation of  $h^d_i$ . It is computed by taking the weighted sum of all the encoder hidden states, each weighted by their dot product with the prior decoder hidden state  $h^d_{i-1}$ .

Image source: Speech and Language Processing. Daniel Jurafsky & James H. Martin. Draft of February 3, 2024.

21

## Encoder vs. Decoder

- **Same Blueprint, Different Wiring**
  - Encoders and decoders are both built on the Transformer architecture.
  - They are **not the same model**
- **The Encoder (The Reader):**
  - Designed for deep context and comprehension.
  - It reads text to build a dense mathematical map of the input
- **The Decoder (The Writer):**
  - Designed for text generation.

22

## Encoder vs. Decoder

| Attribute           | Encoder (e.g., BERT)                                                                 | Decoder (e.g., ChatGPT / GPT-4)                                                   |
|---------------------|--------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------|
| <b>Primary Goal</b> | Context understanding and text analysis.                                             | Autoregressive text generation.                                                   |
| <b>Attention</b>    | Bidirectional: Looks at words to the left and right simultaneously.                  | Causal: Can only look at words to the left (past tokens).                         |
| <b>Ideal Tasks</b>  | Classification, search, sentiment analysis, and NER.                                 | Writing essays, coding, brainstorming, and conversation.                          |
| <b>Analogy</b>      | A professor grading an essay who reads the whole page at once to understand context. | A novelist typing live, where the next word depends strictly on what came before. |

23

## The Summarization Paradox: Processing Without Encoding

| Step                | How ChatGPT (Decoder) Handles Summarization                                                                                                                                                                                                            |
|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1. Input Reading    | <ul style="list-style-type: none"> <li>Reads the source text sequentially from left to right.</li> <li>Because it is a decoder, words at the beginning cannot 'look ahead' to words at the end.</li> <li>Accumulates context incrementally.</li> </ul> |
| 2. The Prompt Pivot | <ul style="list-style-type: none"> <li>Hits the end of your instruction token block (e.g., 'Summarize the text above:').</li> <li>At this point, its internal hidden state holds a highly sophisticated cumulative memory of the text.</li> </ul>      |
| 3. Generation Phase | <ul style="list-style-type: none"> <li>Begins predicting the very first word of the summary based purely on the accumulated prompt memory.</li> <li>It does not compress a global map, it generates forward.</li> </ul>                                |
| 4. Completion Loop  | <ul style="list-style-type: none"> <li>Predicts each subsequent word by appending its own recent output to the context window.</li> <li>It repeats this autoregressive cycle until the text summary is complete.</li> </ul>                            |