CSC-496/696: Natural Language Processing and Text as Data

Lecture 14: RNNs, Encoder-Decoder, and Attention

Patrick Wu

Tuesday, October 22, 2024

Lecture Contents

Announcements

Course Logistics

- Assignment 3 due next Tuesday, October 29 at 11:59pm
- Project proposal due on November 8 at 11:59pm
  - One single-spaced proposal that describes what your research question is, what your data is, and how you plan on answering it
  - Meet with me **now** so we can discuss data and research questions
  - Graded on completion
- In November, I will reach out to everyone to schedule a time to meet about the project just to make sure you are on track

Course Logistics

- The course website's schedule has also been updated
- Quite an extensive reorganizing for the rest of the class
  - We will go right into transformers in the upcoming weeks
  - Rather than discussing many different types of large language models (LLMs), we'll focus on generative LLMs
  - Meta released a set of new LLMs, including ones that we can run on Google Colab directly

Question for the Class

Is anybody not going to use generative LLMs on their final project?

Review of RNNs

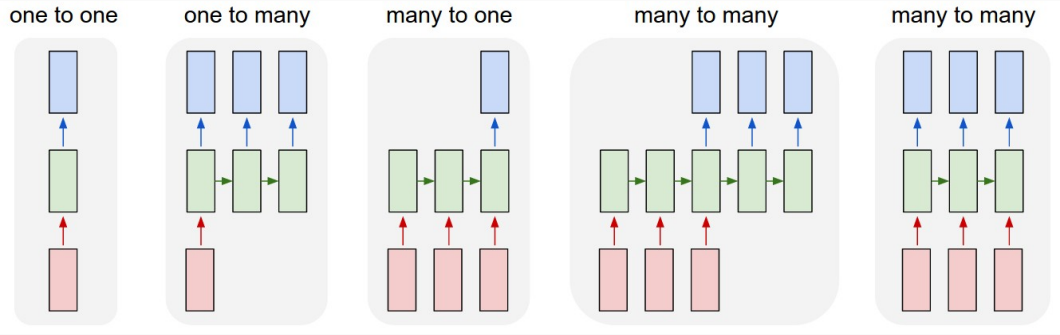
Recurrent Neural Networks

- Fully connected neural network (FCNN): one fixed input, one fixed output
- But language has an aspect of temporality
  - Number of words varies by sentence
  - Similarly, length of documents vary
  - Don't want to create a separate model for each type of document or sentence
- We solve this using a *recurrent neural network* (RNN)

Intuition Behind RNNs

- RNNs as having a *memory mechanism*
- Unlike fully connected neural networks, RNNs can "remember" past information
- They read one word (or character) at a time and have a mechanism to carry information from what they previously encountered

Configurations of RNNs



Review Question 1

What is the primary feature that distinguishes RNNs from FCNN?

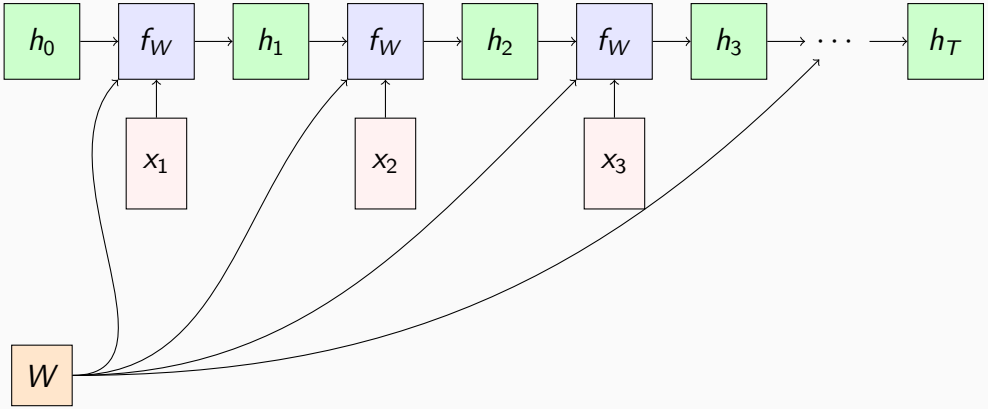
(A) They use multiple layers to extract high-level features from input data

(B) They maintain a memory of previous inputs through internal state loops

(C) They are primarily used for image classification tasks

(D) They have a fixed-length input and output sequences of vectors

Computational Graph of an RNN



Review Question 2

What is a limitation of vanilla RNNs?

(A) They cannot process sequential data

(B) They have too many parameters

(C) They suffer from the vanishing gradient problem (especially with long sequences)

(D) They can only handle fixed-length inputs

Some Code Using RNNs

Code with RNNs

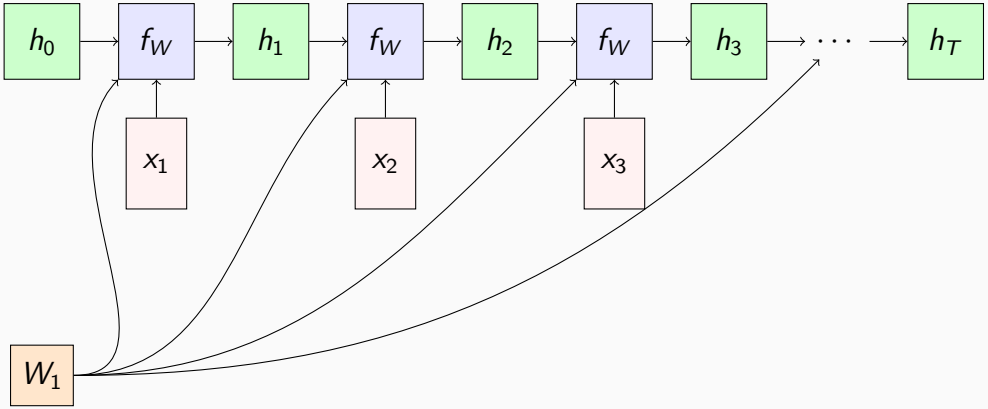
The lab can be found here

Encoder-Decoder Framework

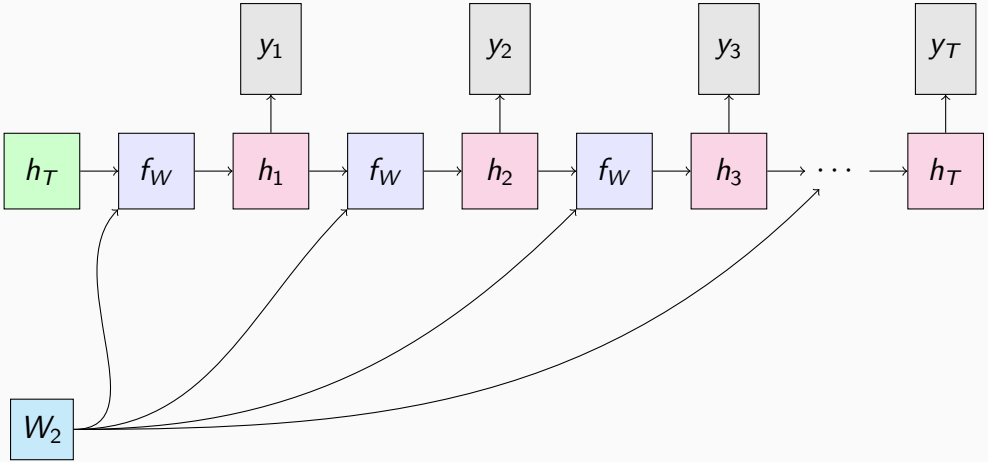
Encoder-Decoder Framework

- Encoder: processes input data and transforms it into a numerical representation
- Decoder: decodes the representation given by the encoder into some kind of output
- Often discussed in the context of RNNs and Transformers, but extends to *any* general model framework where you use a model to encode input information, and then use another model to take as input the output of the encoder and then produce some kind of desired output
- Often used in machine translation, but also used for text summarization, image captioning, etc.

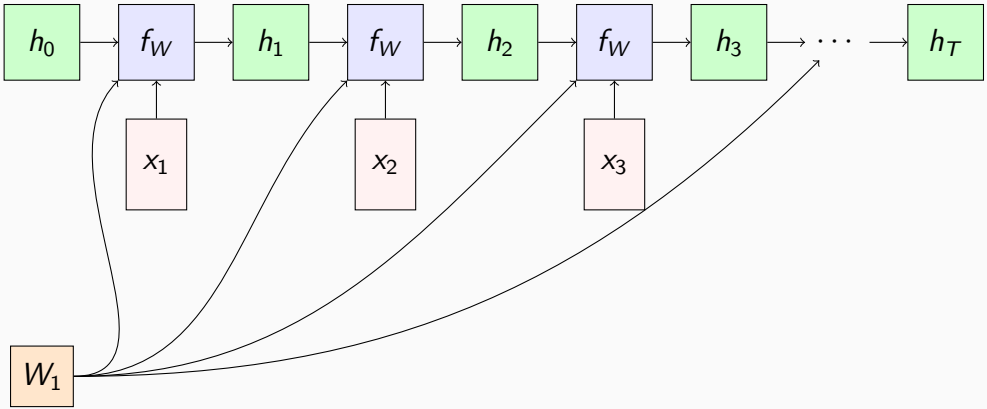
Encoder



Decoder

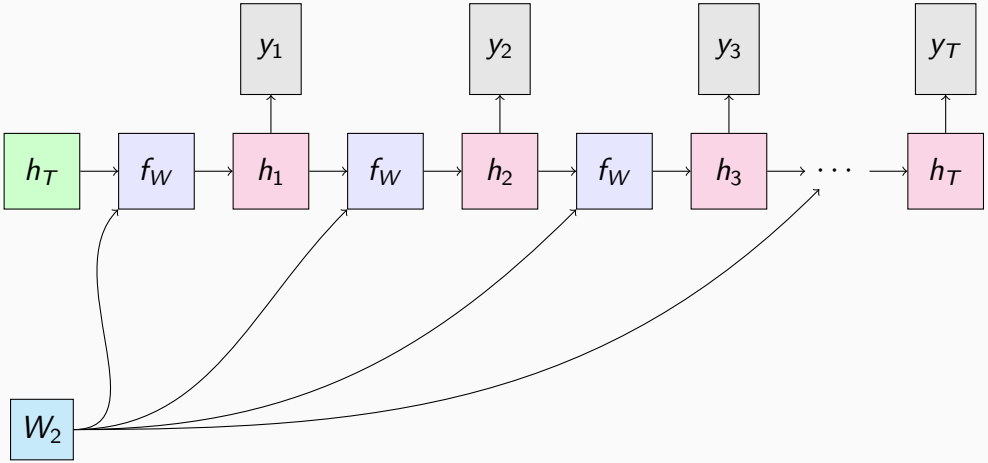


Encoder



h_T is the encoding of the input sequence

Decoder



Then, h_T is used as the initial state and is then *decoded*

Issues with Encoder-Decoder Framework Using RNNs

- Information bottleneck: the entire input sequence is represented by one vector, h_T

Issues with Encoder-Decoder Framework Using RNNs

- Information bottleneck: the entire input sequence is represented by one vector, h_T
- Is a major issue for longer sequences, where information at the start of the sequence might be lost in the process of compressing the sequence to h_T

Issues with Encoder-Decoder Framework Using RNNs

- Information bottleneck: the entire input sequence is represented by one vector, h_T
- Is a major issue for longer sequences, where information at the start of the sequence might be lost in the process of compressing the sequence to h_T
- What if we allowed the decoder to have access to all of the encoder's hidden states, $h_1, \ldots, h_T$?

Issues with Encoder-Decoder Framework Using RNNs

- Information bottleneck: the entire input sequence is represented by one vector, h_T
- Is a major issue for longer sequences, where information at the start of the sequence might be lost in the process of compressing the sequence to h_T
- What if we allowed the decoder to have access to all of the encoder's hidden states, $h_1, \ldots, h_T$?
- Yes! Formalized in the *attention* mechanism

Review Question 3

The encoder model and the decoder model have to be the same in the encoder-decoder framework

(A) True

(B) False

Review Question 4

Why is the information bottleneck an issue for encoders?

Attention

Intuition of Attention

- Developed in 2014 in "Neural Machine Translation by Jointly Learning to Align and Translate" by Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio
  - Paper can be found here
- To improve the encoder-decoder framework using RNNs, we want to allow the decoder to have access to the hidden states of the encoder
- However, using all the states at the same time would create a huge input for the decoder
- Need some mechanism to prioritize which of the encoder states to use at each decoding timestep

Intuition of Attention

- Matches the human intuition of attention as well
- When you read, you do not read every word equally in care
- Some words you read over quickly, while other key words are stored in some kind of "memory"
  - For example, when you read a mystery novel, you often keep track of the clues that appear throughout the story
- When you're reading, you connect previous information to what you're currently reading, but you don't weight all previous information in the same way
  - Mystery novel: you remember the major themes and clues, but you don't need to remember what led to those clues or how those clues were found

The Attention Mechanism Operationalizes this Intuition

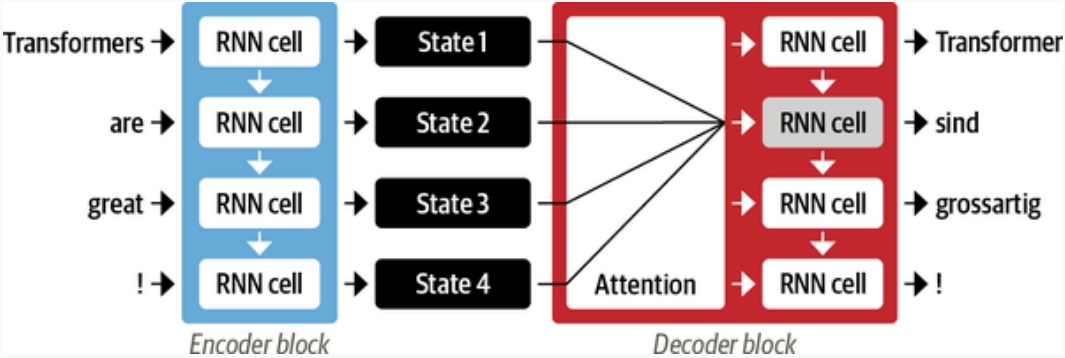
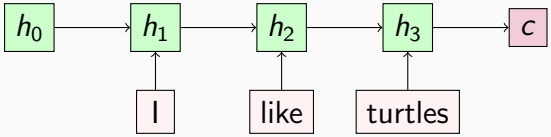


Figure taken from Tunstall, von Werra, and Wolf (2022).

Intuition of Attention

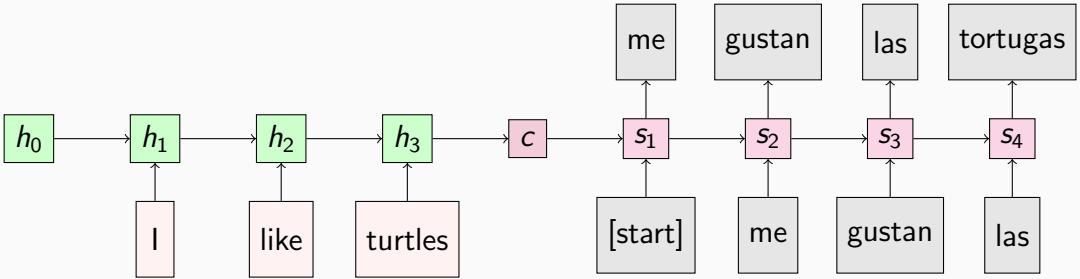
- The attention mechanism prioritizes which of the previous states to use at each decoder step
- In other words, it allows the decoder to assign a different weight (or "attention") to each of the encoder states at ever decoding timestep
- These attention-based models can then learn alignments between the output and the input
- First applied to machine translation, which saw dramatic improvements when using attention

Encoder

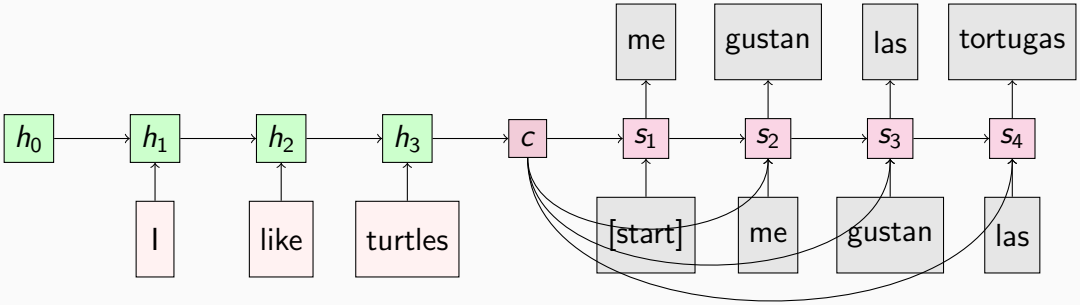


We use the encoder to predict a "context" embedding, c. Each h_t is calculated as $h_t = f_W(x_t, h_{t-1})$.

Encoder-Decoder



Encoder-Decoder with Context Sharing



$$h_t = f_W(x_t, h_{t-1})$$

$$s_t = g_U(y_{t-1}, s_{t-1}, c)$$

c is the bottleneck

- Input sequence ("I like turtles") is bottlenecked through the vector c. If T was *much* longer, this would present a major issue
- Even if c is made available for each decoder step, c still contains a lot of information about the sequence
- We want to allow decoder steps to access each h_t
- Attention mechanism: a way to allow the decoder to get information from all hidden states of the encoder, not just the last hidden state

Dot-Product Attention

- We measure how similar the decoder hidden state is to an encoder hidden state

Dot-Product Attention

- We measure how similar the decoder hidden state is to an encoder hidden state
- Recall that

$$s_t = g_U(y_{t-1}, s_{t-1}, c)$$

Dot-Product Attention

- We measure how similar the decoder hidden state is to an encoder hidden state
- Recall that

$$s_t = g_U(y_{t-1}, s_{t-1}, c)$$

- What we'll do is calculate a different c for each decoder hidden state

Dot-Product Attention

- We measure how similar the decoder hidden state is to an encoder hidden state
- Recall that

$$s_t = g_U(y_{t-1}, s_{t-1}, c)$$

- What we'll do is calculate a different c for each decoder hidden state
- To do this, we'll calculate a score between s_{t-1} and all encoder hidden states, h_j

Dot-Product Attention

- We measure how similar the decoder hidden state is to an encoder hidden state
- Recall that

$$s_t = g_U(y_{t-1}, s_{t-1}, c)$$

- What we'll do is calculate a different c for each decoder hidden state
- To do this, we'll calculate a score between s_{t-1} and all encoder hidden states, h_j
- We'll use the dot product:

$$\text{score}(s_{t-1}, h_j) = s_{t-1} \cdot h_j$$

We repeat this for every encoder hidden state

Dot-Product Attention

After calculating this score for all encoder hidden states, we can then calculate the weights using the softmax

$$\alpha_{tj} = \text{softmax}(\text{score}(s_{t-1}, h_j)) = \frac{\exp(\text{score}(s_{t-1}, h_j)}{\sum_k \exp(\text{score}(s_{t-1}, h_j))}$$

α_{tj} is the weight between the decoder hidden state s_{t-1} and encoder hidden state h_j

Dot-Product Attention

Finally,

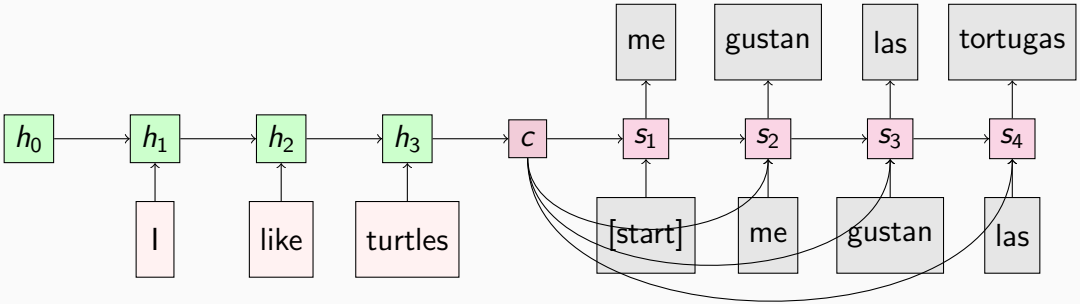
$$c_t = \sum_k \alpha_{tk} h_k$$

We can then calculate s_t as follows

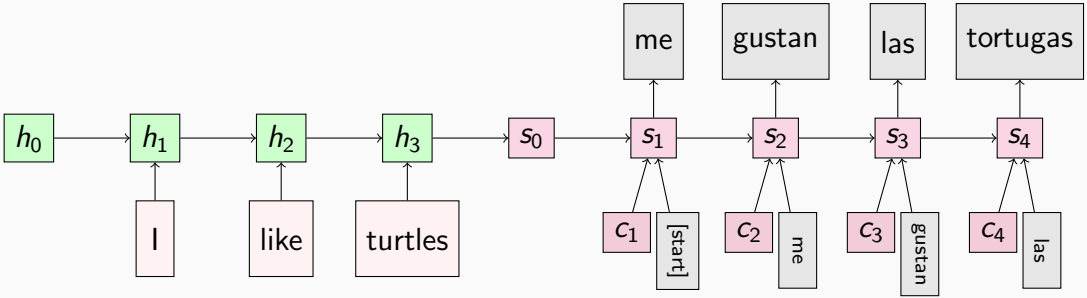
$$s_t = g_U(y_{t-1}, s_{t-1}, c_t)$$

Notice that c_t is a weighted sum of *all* of the hidden state of the encoder. By backpropping through everything, we can learn which encoder states to pay more "attention" to at each decoding step.

Encoder-Decoder Framework without Dot-Product Attention



Encoder-Decoder Framework with Dot-Product Attention



Alternative Scoring for Attention Mechanism

- There are more sophisticated scoring functions for the attention mechanism
- For example,

$$\text{score}(s_{t-1}, h_j) = s_{t-1} W_s h_j$$

  The weights W_s can be learned during normal end-to-end training, and can model more complex similarities
- Dot-product attention is often used for its simplicity

Concluding Lecture

Attention

- Attention allows us to *attend* to the previous encoder hidden states
- This approach saw major improvements in tasks like machine translation, image captioning, etc.
- One drawback still: inputs were inherently sequential, so each input had to be processed one token at a time
- In "Attention is All You Need," one of the most famous papers in NLP, the authors proposed an approach that uses attention without recurrence, developing an architecture called the Transformer (you can read the paper here)