EECS 182 Deep Neural Networks
Fall 2022 Anant Sahai

Discussion 7

1. Attention Mechanisms for Sequence Modelling

Sequence-to-Sequence is a powerful paradigm of formulating machine learning problems. Broadly, as long as we can formulate a problem as a mapping from a sequence of inputs to a sequence of outputs, we can use sequence-to-sequence models to solve it. For example, in machine translation, we can formulate the problem as a mapping from a sequence of words in one language to a sequence of words in another language. While some RNN architectures we previously covered possess the capability to maintain a memory of the previous inputs/outputs, to compute output, the memory states need to encompass information of many previous states, which can be difficult especially when performing tasks with long-term dependencies.

To understand the limitations of vanilla RNN architectures, we consider the task of changing the case of a sentence, given a prompt token. For example, given a mixed case sequence like "<U> I am a student", the model should identify this as an upper-case task based on token <U>, and convert it to "I AM A STUDENT". Similarly, given "<L> I am a student", the lower-case task is to convert it to "i am a student".

We can formulate this task as a character-level sequence-to-sequence problem, where the input sequence is the mixed case sentence, and the output sequence is the desired case sentence. In this exercise, we use an encoder-decoder architecture to solve the task. The encoder is a vanilla RNN that takes the input sequence as input, and outputs a sequence of hidden states. The decoder is also a vanilla RNN that takes the last hidden state from the encoder as input, and outputs the desired case sentence.

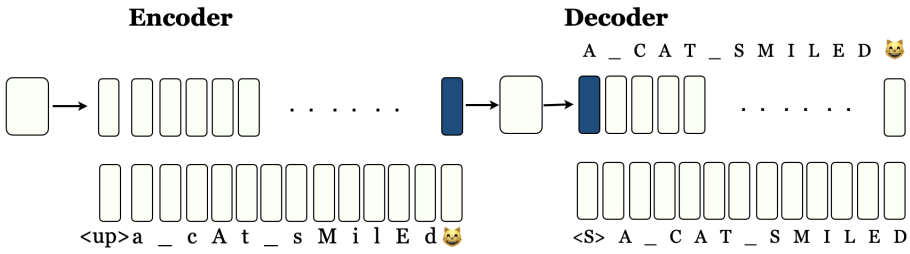

Figure 1: String Manipulation as a Sequence-to-Sequence Problem

(a) What information do RNNs store?

It is important to understand how information propagates through RNNs. Particularly in the context of sequence-to-sequence models, we want to understand what information is stored in the hidden states, and what information is stored in the weights (encoder & decoder). To understand this, we consider the different components of the RNN architecture.

- **Input sequence**: The input sequence is a sequence of T tokens.
- **Encoder Weights**: The shared learnable parameters of the encoder, $W_{\text{enc}} \in \mathbb{R}^{d \times h}$
- **Bottleneck Activations**: The encoder hidden state at time T, that is passes to decoder.
- **Output sequence**: The output sequence is a sequence of T vectors (might be different length).
- **Decoder Weights**: The shared learnable parameters of the decoder, $W_{\text{dec}} \in \mathbb{R}^{d \times h}$

Consider the following questions in the context of these modules:

i. Which of these components change during inference?
ii. When performing gradient based updates, how are the decoder weights trained? How is gradient propagated through the encoder?
iii. What is the qualitative nature of information captured by the weights?
iv. During training, what is the role of the input/output sequence?

2.5in

(b) Information Bottleneck in RNNs

Consider the architecture shown in Figure 1. This is a simple encoder-decoder architecture with a single hidden layer in the encoder and decoder. The encoder takes the input sequence as input, and outputs a sequence of hidden states. The decoder takes the last hidden state from the encoder as input, and outputs the desired case sentence. What information needs to be stored in the hidden state to perform the upper-case/lower-case task? Are there any limitations to this architecture?

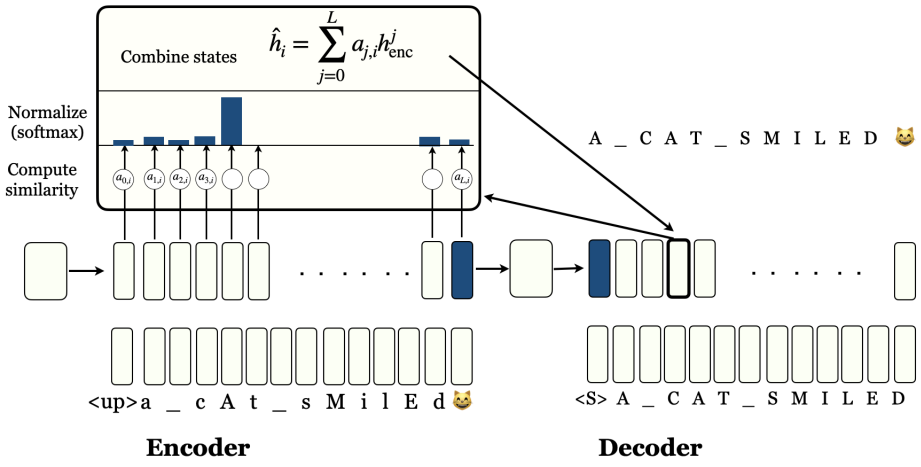

Figure 2: Attention Mechanism for Sequence Modelling with RNNs.

2.5in

(c) Introducing Attention

Instead of storing all the information in the hidden state, we can use attention to selectively store information. The idea of attention is to query the encoder hidden states with a query vector, and use the resulting attention weights to compute a weighted sum of the hidden states. This weighted sum is then used as the input to the readout layer that computes the output token at each time step of the decoder. How would you modify the encoder-decoder architecture to incorporate attention?

2.5in

(d) Attention & RNNs

How does adding attention allow the model to bypass the information bottleneck? In particular for the capitalization task, what information in the following modules would allow the model to perform the task?

- **Encoder Weights**
- **Attention Scores**
- **Bottleneck Activations**

- **Decoder Weights**

1.5in

(e) Positional Encoding

As noted above, the hidden state of the encoder at position t should contain information about the position of token in the input sequence. To incorporate this information, we can add a *positional encoding* to the input tokens. For sequences of length T discuss how you would add positional encoding to the input sequence.

2in

(f) Prompt-Engineering : Alternative Designs

Based on our understanding of challenges with modelling long-range dependencies, a potential bottleneck in the previous design is that the prompt token is passed at the beginning of the input-sequence to the encoder. This means that the encoder bottleneck activation has to store information about the prompt token, and the entire input sequence. Are there any alternative ways of tokenizing the sequences? Discuss the implications for both vanilla Encoder-Decoder models, and models with attention.

2.5in

(g) Limits of Expressivity

Let's consider a different seq-to-seq task, where given a sentence with arbitrary number of spaces, we want to remove all the spaces. For example, given the sentence "I am a student", we want to output "Iamastudent". Informally, using the same architecture with attention as before, could we solve this task? Are there any simple modifications that could resolve this limitation? *(Hint : Think about modifying the output vocabulary.)*

2.5in

References

Baldi, P. and Hornik, K. Neural networks and principal component analysis: Learning from examples without local minima. *Neural networks*, 2(1):53–58, 1989.

Luong, M.-T., Pham, H., and Manning, C. D. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 1412–1421, 2015.

Tsai, Y.-H. H., Bai, S., Yamada, M., Morency, L.-P., and Salakhutdinov, R. Transformer dissection: An unified understanding for transformer's attention via the lens of kernel. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 4344–4353, 2019.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Contributors:

- Kumar Krishna Agrawal.

