EECS 182 Deep Neural Networks
Fall 2026 Gireeja Ranade

Discussion 10

1. Written Coding: A Selective SSM and the Parallel Scan

A linear time-invariant (LTI) state-space model applies the same dynamics to every token, so it cannot decide, based on content, which tokens to keep. Selective state-space models (as in Mamba) let the step size depend on the current input. Take the continuous-time system $\dot{h} = -h + u$ in each of d channels. Zero-order hold with an input-dependent step size $\Delta_t = \mathrm{softplus}(z_t) = \ln(1 + e^{z_t})$, where $z_t = w \odot u_t + c$, gives

$$h_t = g_t \odot h_{t-1} + q_t, \qquad g_t = e^{-\Delta_t}, \qquad q_t = (1 - g_t) \odot u_t,$$

with $h_0 = 0$. The code below computes all of $h_1, \ldots, h_T$ (rows $0, \ldots, T-1$ of the output) in two ways.

```
import torch

def selective_gate(u, w, c): # u: [T, d]; w, c: [d]
    z = u * w + c # [T, d]
    g = torch.sigmoid(-z) # [T, d]
    q = _______1_______ # [T, d]
    return g, q

def scan_sequential(g, q): # g, q: [T, d]
    h = torch.zeros_like(g[0]) # h_0 = 0
    hs = []
    for t in range(g.shape[0]):
        h = _______2_______ # [d]
        hs.append(h)
    return torch.stack(hs) # [T, d]

def combine(g1, q1, g2, q2): # apply (g1, q1) first, then (g2, q2)
    return _______3_______, _______4_______

def scan_parallel(g, q):
    g, q = g.clone(), q.clone()
    T, step = g.shape[0], 1
    while step < T:
        g_prev, q_prev = torch.ones_like(g), torch.zeros_like(q)
        g_prev[step:], q_prev[step:] = g[:-step], q[:-step]
        g, q = combine(g_prev, q_prev, g, q)
        step *= 2
    return _______5_______ # [T, d]
```

 All Rights Reserved. This may not be publicly shared without explicit permission.

(a) Line 5 computes the gate with `torch.sigmoid(-z)`. **Show that this equals $e^{-\Delta_t}$, and fill in blank 1.**

(b) **Fill in blank 2. What does one step do when $z_t \gg 0$, and when $z_t \ll 0$?**

(c) Because g_t changes with t, the layer is not a convolution with one fixed kernel, so the FFT trick from lecture no longer applies. But each step is the affine map $h \mapsto g_t \odot h + q_t$. **Fill in blanks 3 and 4** so that `combine` returns the pair for "first (g_1, q_1), then (g_2, q_2)". **Why does it matter that this composition is associative?**

(d) After the loop in `scan_parallel`, row t holds the composition of all steps up to t. **Fill in blank 5. How many times does the loop run?**

2. 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

 All Rights Reserved. This may not be publicly shared without explicit permission.

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.

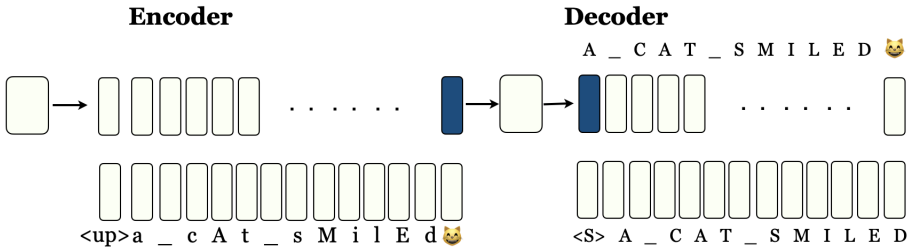


Figure 1: String Manipulation as a Sequence-to-Sequence Problem

(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_{enc}
- **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_{dec}

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. During training, what is the role of the input/output sequence?

 All Rights Reserved. This may not be publicly shared without explicit permission.

(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?**

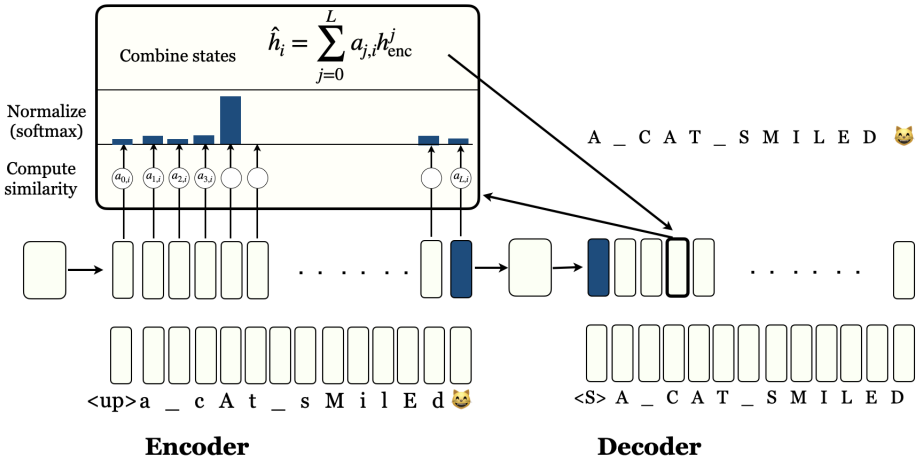


Figure 2: Attention Mechanism for Sequence Modelling with RNNs.

(c) Attention & RNNs

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

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

Contributors: Matteo Guarrera, Sizhe Chen, Mert Cemri, Gireeja Ranade, Kumar Krishna Agrawal.

 All Rights Reserved. This may not be publicly shared without explicit permission.