Sequence-to-sequence

Language modeling

$p(x_t \mid x_{t-1}, \ldots, x_1)$

How to train? Collect $x_1, x_2, \ldots, x_T$

"Teacher forcing": Feed in sequence upto x_t, train model to predict x_t

Sequence classification

$p(y \mid x_1, \ldots, x_T)$

$h_t = f(\underbrace{x_t, h_{t-1}}_{\text{RNN}})$

$o = W_o h_T + b_o$

↑ scores

Sequence-to-sequence

$$p(y_t \mid \underbrace{y_{t-1}, y_{t-2}, \cdots, y_1}_{\text{previous output}}, \underbrace{x_T, x_{T-1}, \cdots, x_1}_{\text{entire input}})$$

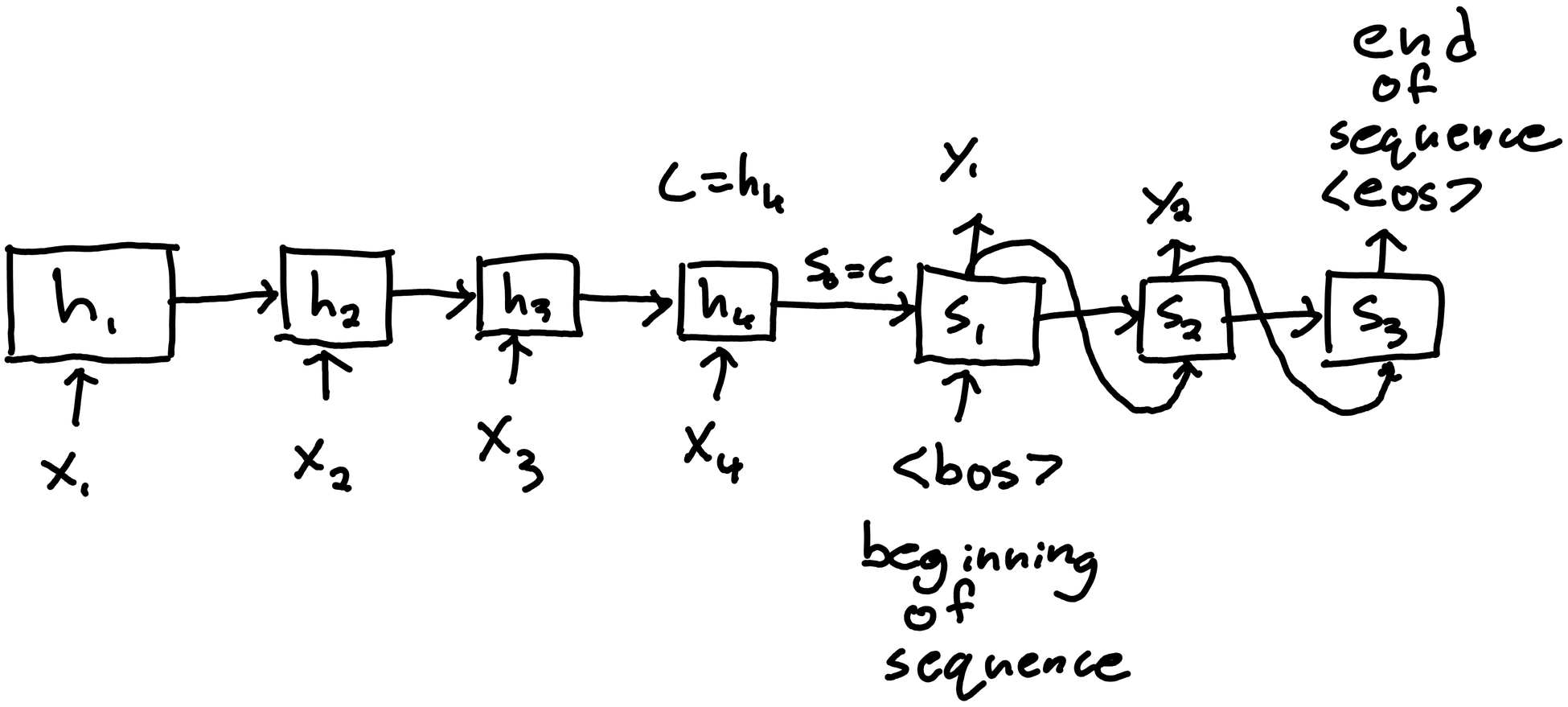
y_t: current output

ex Machine translation, x = English sentence
y = French translation of x

How to build a seq2seq model with RNNs:

1. $h_t = f(x_t, h_{t-1})$ "encoder" RNN
2. $c = h_T$ "context"
3. $s_t = g(y_{t-1}, s_{t-1}, c)$ "decoder" RNN
   ↳ used to predict y_t

How to make g a function of c?
e.g. $s_0 = c$ or concat(embed(y_{t-1}), c)

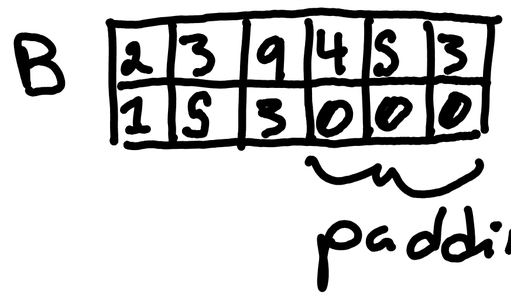

h_1
h_2
h_3
h_4
$c = h_4$
$s_0 = c$
s_1
s_2
s_3
y_1
y_2
end of sequence <eos>
x_1
x_2
x_3
x_4
<bos>
beginning of sequence

Padding

Batch of sequences has shape $B \times T$ (batch size, sequence length)

→ all sequences must be length-T!

Use padding:



① is special token (like <bos>)

For encoder, padding token → ignore input, set $h_t = h_{t-1}$

For decoder, padding token → don't apply loss

How to generate outputs?

1. Autoregressive sampling

$$y_{t'} \sim p(y_{t'} \mid y_{t'-1}, \cdots, y_1, x_T, \cdots, x_1)$$

$$y_{t'+1} \sim p(y_{t'+1} \mid y_{t'}, \cdots, y_1, x_T, \cdots, x_1)$$

2. Greedy sampling:

$$y_{t'} = \text{argmax}[p(y_{t'} \mid y_{t'-1}, \cdots, y_1, x_T, \cdots, x_1)]$$

3. Top-k sampling:

$$y_{t'} \sim \text{top-k}[p(y_{t'} \mid y_{t'-1}, \cdots y_1, x_T, \cdots x_1)]$$

4. Top-p sampling (Nucleus sampling)
   Only sample the tokens that make up p% of the total probability mass

5. Beam search: Consider the k most probable prefixes at each timestep.

Attention

Read-only associative array / key-value store

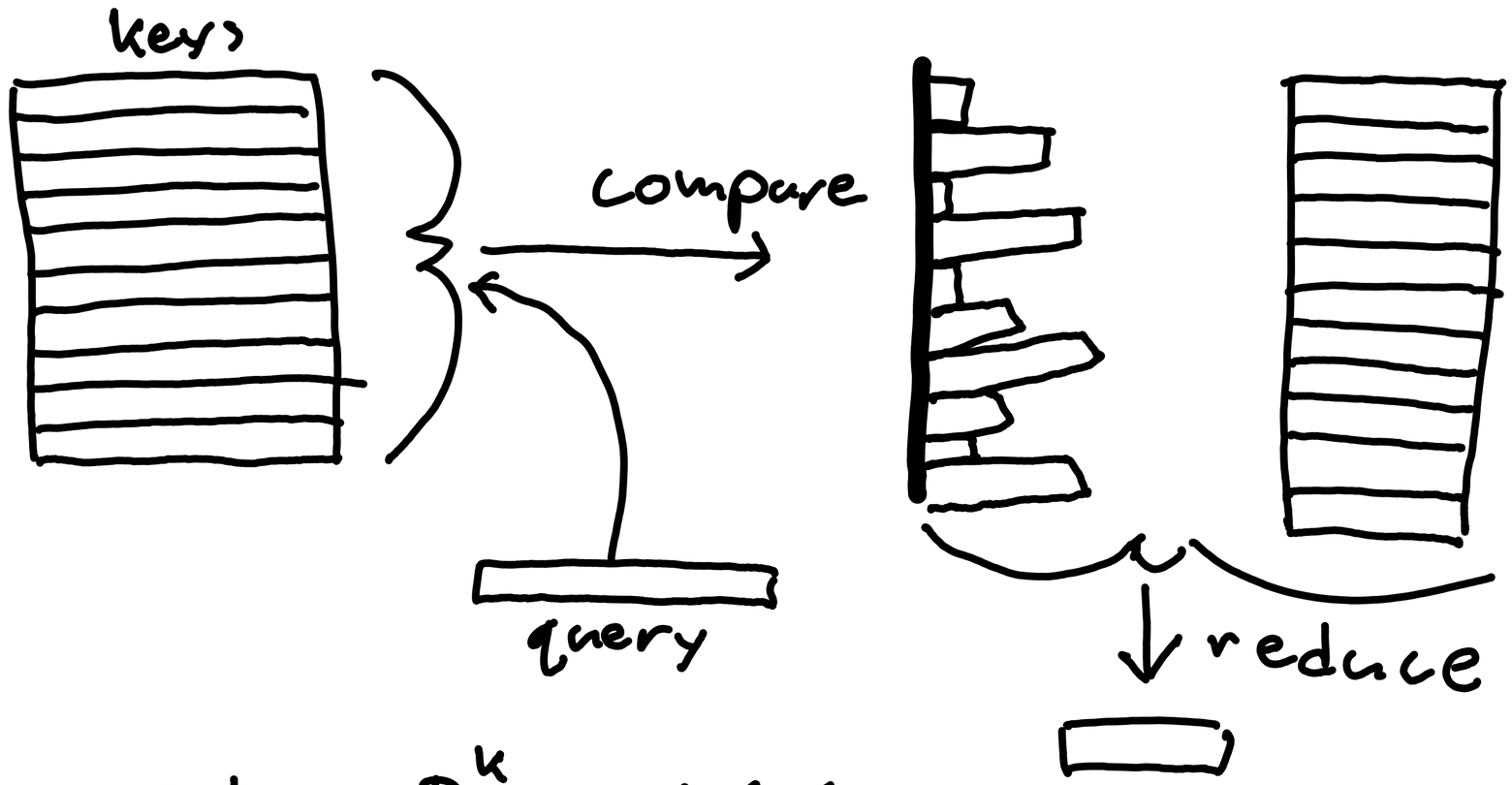
- Values: set of things we're storing
- Keys: identifiers, one for each value
- Query: identifier of the thing to retrieve

Implements two functions:

- Comparison function: determines how well the query matches a given key $\alpha(k_i, q)$
- Reduction function: Given results from the comparison function, return something from values

ex/ dict. Comparison function tests for equality, reduction function returns the value for matching key.

"Soft" (differentiable) associative memory
aka attention mechanism



keys: $k_i \in \mathbb{R}^k$, $i \in \{1, \dots, N\}$

query: $q \in \mathbb{R}^k$

values: $v_i \in \mathbb{R}^v$, $i \in \{1, \dots, N\}$

comparison: $a_i = \alpha(q, k_i) \in \mathbb{R}$, $a \in \mathbb{R}^N$

Reduction: $f(\{v_1, \dots, v_N\}, a) \in \mathbb{R}^v$

ex/ Soft 1-nearest neighbor classifier

query: data point we want to classify

keys: datapoints from the training set

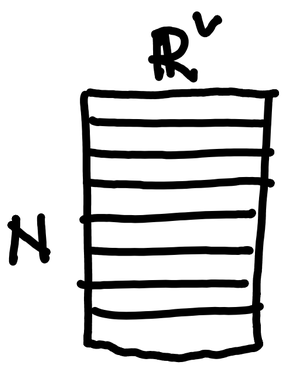
values: labels of datapoints from the training set

$$a(q, k_i) = -\|q - k_i\| = a_i$$

$$f(\{v_1, \dots, v_N\}, q) = \text{softmax}(a) \cdot V$$

↓

dot product

$V =$ 

if we replace softmax with arg max, just a normal 1-nn classifier

Sequence-to-sequence with attention

If $c = h_T$, then h_T has to encode all relevant information about x.

Decoder output at some time step may depend on some specific inputs.

Rather than having a single c, let's have a different $c_{t'}$ at each output timestep.

query: $s_{t'-1}$ (decoder state from previous timestep)

values: $h_1, h_2 \ldots, h_T$ (encoder states)

keys: $h_1, h_2, \ldots, h_T$

$$\alpha(q, k_i) = \underbrace{w_v^T}_{\text{learnable parameters}} \tanh(w_q q + \underbrace{w_k}_{\text{learnable parameters}} k_i) = a_i$$

$$c_{t'} = \text{softmax}(a) \cdot [h_1, h_2 \ldots, h_T]$$

use $c_{t'}$ as an additional input to the decoder RNN.

$$s_{t'} = g(s_{t'-1}, y_{t'-1}, c_{t'})$$

Self-attention

In self-attention, the queries, keys, and values all come from the same sequence.

$$x_1, x_2, \dots x_N, \quad x_i \in \mathbb{R}^m$$

queries: $W_Q x_1, W_Q x_2, \dots, W_Q x_N = q_1, \dots, q_N \in \mathbb{R}^d$

keys: $W_k x_1, W_k x_2, \dots, W_k x_N = k_1, \dots, k_N \in \mathbb{R}^d$

values: $W_v x_1, W_v x_2, \dots, W_v x_N = v_1, \dots, v_N \in \mathbb{R}^v$

$\alpha(q, k_i) = q^T k_i / \sqrt{d}$ (if q and k_i are $\mathcal{N}(0, I)$ then $q^T k_i / \sqrt{d}$ will be too)

$f(\{v_1, \dots, v_N\}, a) = \text{softmax}(a) \cdot v$

length-N sequence $\rightarrow$ length-N sequence of weighted averages of the (projected) input

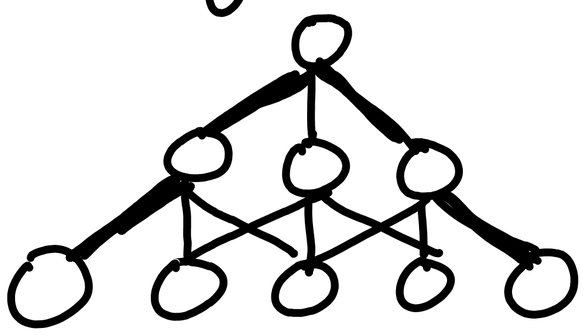
"Path length"

How many operations are required to "mix" information between two points in the input?

$$x = [x_1, x_2, \ldots, x_6]$$

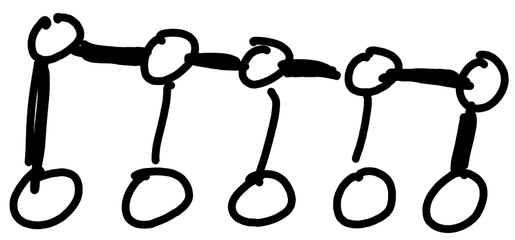
For different models, what is the path length between x_1 and x_5

ConvNet
length-3 kernels



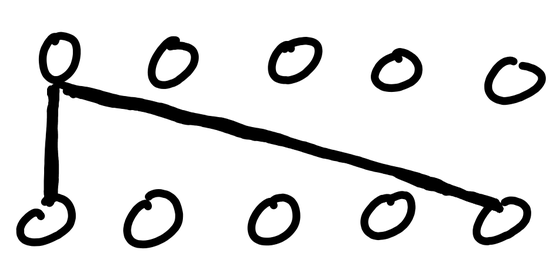
Path length 2

RNN



Path length 5

Self-attention



Path length 2

-