4. Sequence Generation

Sequence Generation

Text classification: $h : \nu^n \to \{0, ..., K\}$

Sequence generation: $h : \nu_{\text{in}}^n \to \nu_{\text{out}}^m$

- Summarization: document to summary
- Open-domain dialogue: context to response
- Parsing: sentence to linearized trees
- In general: text to text

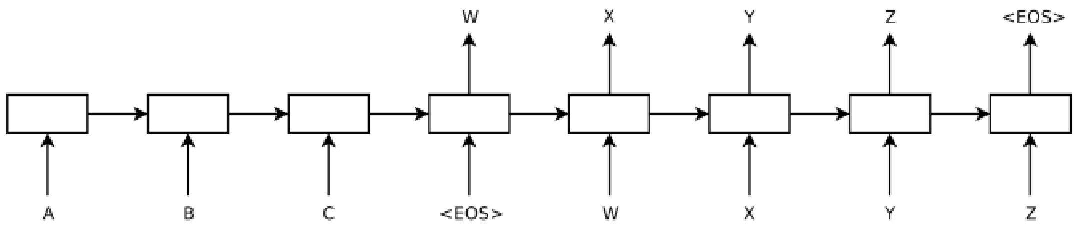
Main difference (and challenge) is that the output space is much larger.

Reduce generation to classification

We want to model the next word distribution $p(y_i | y_{<i}, x)$.

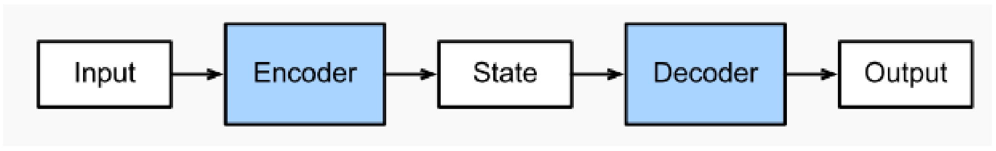
- Input: a sequence of tokens (prefix and input)
- Output: the next word from the output vocabulary

We can use an RNN to model $p(y_i | y_{<i}, x)$.



EOS stands for "End of Sequence”.

The encoder-decoder architecture



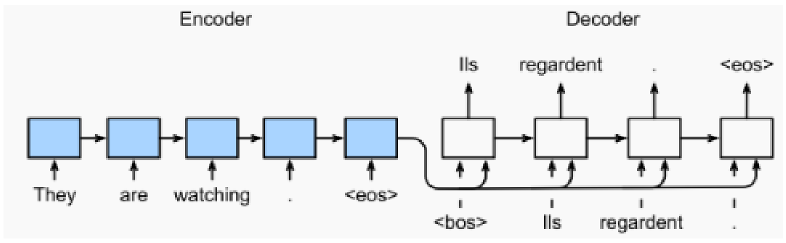
The encoder reads the input:

$$\text{Encoder}(x_1, ..., x_n) = [h_1, ..., h_n]$$

The decoder writes the output:

$$\text{Decoder}(h_1, ..., h_n) = [y_1, ..., y_m]$$

RNN encoder-decoder model



The encoder embeds the input recurrently and produce a context vector:

$$h_t = \mathrm{RNNEncoder}(x_t, h_{t-1}), \quad c = f(h_1, ..., h_n)$$

The decoder produce the output state recurrently and map it to a distribution over tokens:

$$s_t = \mathrm{RNNDecoder}([y_{t-1}; c], s_{t-1}), \quad p(y_t | y_{<t}, c) = \mathrm{softmax}(\mathrm{Linear}(s_t))$$

Bi-directional RNN encoder: each hidden state should summarize both left and right context

- Use two RNNs, one encode from left to right, the other from right to left
- Concatenate hidden states from the two RNNs

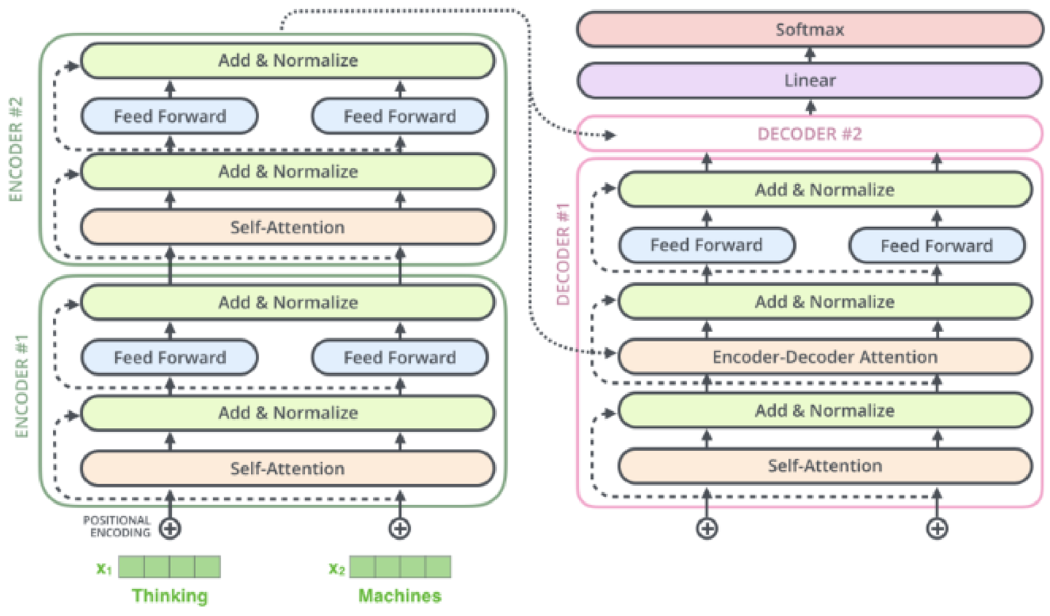
Multilayer RNN (Typically 2–4 layers)

- Improve model capacity (scaling up)
- Inputs to layer 1 are words, inputs to layer l are outputs from layer $l-1$

Encoder-decoder attention

Motivation: should we use the same context vector for each decoding step? We may want to “look at” different parts of the input during decoding.

Transformer encoder decoder model



- Stack the tranformer block (typically 12–24 layers)
- Decoder has an additional encoder-decoder multi-head attention layer

Training

Maximum likelihood estimation

$\max \sum_{(x,y)\in D} \sum_{j=1}^{m} \log p(y_j \mid y_{<j}, x; \theta)$

What should be the prefix $y_{<j}$?

Option 1: whatever generated by the model

Option 2: the groundtruth prefix (teacher forcing)

Decoder attention masking

The output of self-attention depends on all tokens $y_1, \ldots y_m$.

But the decoder is supposed to model $p(y_t \mid y_{<t}, x)$, without looking at the “future” $(y_{t+1}, \ldots, y_m)$!

How to fix it? Practically, set $a(s_i, s_j)$ to $-\inf$ for all $j > i$ and for $i = 1, \ldots, m$.

- The attention matrix is a lower-triangular matrix.

Inference

The encoder-decoder model defines a probability distribution $p(y|x;\theta)$ over sequences. How to generate sequences?

Argmax decoding: return the most likely sequence

$\hat{y} = \arg\max_{y\in\nu_{\text{out}}^{n}} p(y|x;\theta)$

- but exact search is intractable

Greedy decoding: return the most likely symbol at each step

$\hat{y} = \arg\max_{y \in \nu_{\text{out}}} p(x, y_{<t} | \theta)$

Beam search: maintain k (beam size) highest-scored partial solutions at every step

- At each step, rank symbols by log probability of the partial sequence
- Keep the top-k symbol out of all possible continuations
- Save backpointer to the previous state

Sampling-based decoding: If we have learned a perfect $p(y|x)$, shouldn't we just sample from it?

- While output is not EOS:
  - Sample next word from $p(\cdot\,|\text{prefix}, \text{input}; \theta)$
  - append the word to prefix

Tempered sampling: concentrate probability mass on highly likely sequences

- Scale scores before the softmax layer, making the distribution more peaky
  $\exp(\text{score}(w)/T)$ where typically $T \in (0, 1)$

Truncated sampling: truncate the tail of the distribution

- Top-k sampling:
  - Rank all tokens $w \in \nu$ by $p(y_t = w | y_{<t}, x)$
  - Only keep the top k of those and renormalize the distribution
- Top-p sampling:
  - Rank all tokens $w \in \nu$ by $p(y_t = w | y_{<t}, x)$
  - Only keep the tokens in the top p probability mass and renormalize the distribution

Decoding in practice

- Can combine different tricks (e.g., temperature + beam search, temperature + top-k)
- Use beam search with small beam size for tasks where there exists a correct answer, e.g. machine translation, summarization
- Use top-k or top-p for open-ended generation, e.g. story generation, chit-chat dialogue, continuation from a prompt
- As models getting better/larger, sampling-based methods tend to work better