Sequence-to-Sequence Models

Sequence-to-Sequence models map a source sequence to a target sequence, both of **arbitrary lengths** - differs from **sequence tagging**, where the input and output sequences have the **same length**.

Example - **Machine Translation**: Translate a **source sentence** x in one language to a **target sentence** y in another language.

Statistical Machine Translation

- **Goal**: Given a **source sentence** x, find the **most likely target sentence** y; $\hat{y} = argmax_y p(y|x)$;
- **Key idea**: use **Bayes' rule** to **invert** the **conditional probability** $p(y|x)$;
- **Translation model**: models how words and phrases are **translated** from one language to another - learn from **parallel data**;
  - $p(x, a|y)$ - **source sentence** x, **alignment** a and **target sentence** y - word alignments are word-to-word correspondences between the source and target sentences;
  - Alignment can be one-to-many (**word fertility**);
  - or many-to-may (**phrase based**);
- **Language model**: models the **target language** - learn from **monolingual data**;
- To search the best translation, we need to solve $\hat{y} = argmax_y \sum_a P(x, a|y)P(y)$, combining the **translation model** and the **language model**;
  - A typical approach is to use **heuristic search** to gradually build the translation, discarding hypotheses that are unlikely.

Neural Machine Translation

- **Neural Machine Translation (NMT)** is a **sequence-to-sequence** model for **machine translation** - machine translation with a **single neural network**;
- End-to-end training with parallel data;
- The underlying architecture is an **encoder-decoder**, also known as **sequence-to-sequence** model;
- **Encoder**: encodes the **source sentence** into a **fixed-length vector state** - $c = RNN(x)$;
  - The probability of sequence $y|x$ is $\sim RNNLM(c)$;
- **Decoder**: generates the **target sentence** from the **state vector**.

Beam Search

- **Beam search** is a **heuristic search** algorithm that explores a **graph** by expanding the most promising node in a limited set - **beam size** k - beam sizes vary from 4 to 12;
- **Approximate search** strategy for **decoding** in **sequence-to-sequence** models;
- At each decoder step, keep track of the k **most likely** partial translations;
- For $k = 1$, beam search is **greedy search**.

Problem: sentences are of **variable length**, but vectors are of the same size;

Solution: use **matrices** instead of **vectors**:

- Fixed number of rows, but variable number of columns;
- Before generating each word in the decoder, use an **attention mechanism** to **focus** on the **relevant parts** of the **source sentence**.

Encoder-Decoder with Attention

- Strategies to encode a sentence as a matrix:
  - CNNs;
  - **Bidirectional LSTMs**;
  - Transformer networks.
- We now have a matrix F representing the input; How to generate from it?
  - **Attention mechanism - focus** on the **relevant parts** of the **source sentence**.

Attention Mechanism

- Generate the output sentence **word-by-word** using an RNN;
- At each output position t, the RNN receives two inputs:
  - The **previous word** y_{t-1} - fixed-size vector embedding;
  - A fixed-size vector encoding a view of the input matrix F - **context vector** F_{a_t};
- The input columns weighting at each time-step (a_t) is called the **attention distribution**.

Algorithm:
Let $s_1, s_2, ..., s_T$ be the hidden states of the encoder RNN.
When predicting the t-th word:

1. Compute similarity with each of the source words: $z_{t,i} = v^T g(Wh_i + Us_{t-1} + b), for i = 1, ..., T$;
2. From $z_t = (z_{t,1}, ..., z_{t,L})$, compute the **attention distribution**: $a_t = softmax(z_t)$;
3. Use attention to compute input conditioning state $c_t = F_{a_t}$;
4. Update RNN state $s_t = RNN(s_{t-1}, y_{t-1}, c_t)$;
5. Predict next word: $y_t \sim p(y_t|s_t)$.