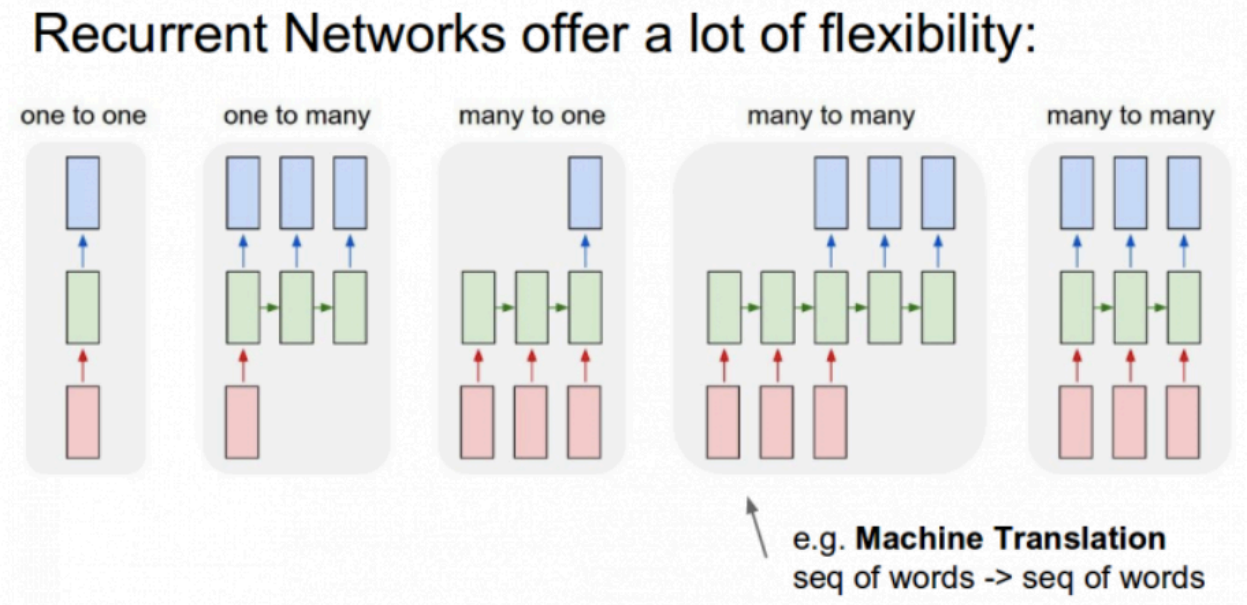
Sequence2Sequence Model

When to use a seq2seq model?

Suppose you have a series of statements :=

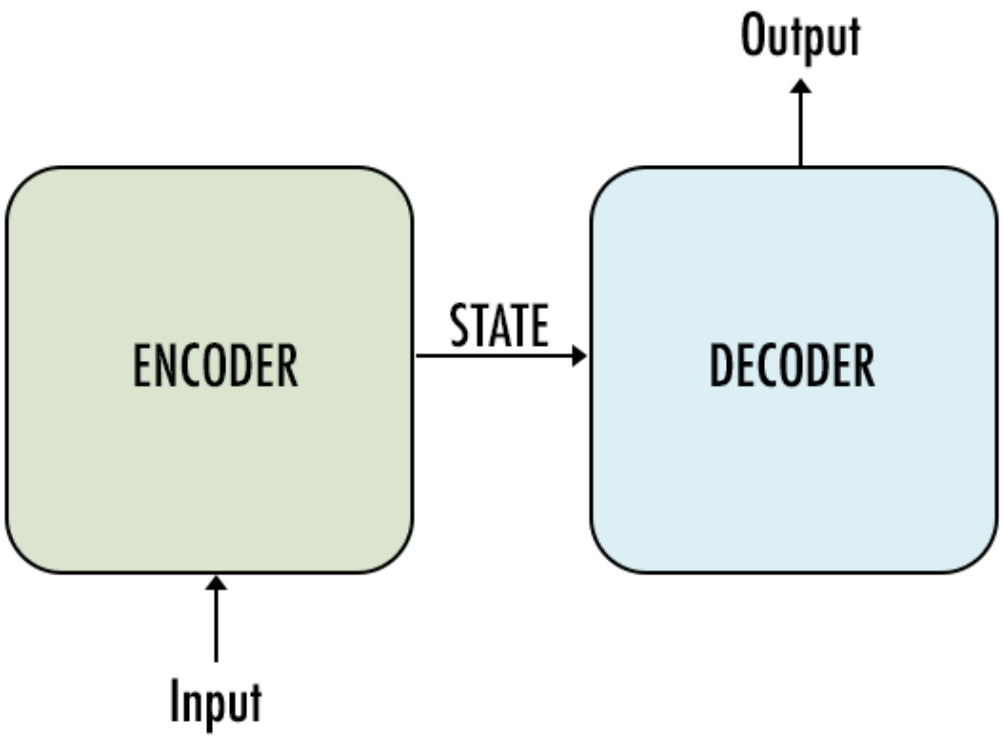
Joe went to the kitchen. Fred went to the kitchen. Joe picked up the milk. Joe travelled to the office. Joe left the milk. Joe went to the bathroom.

This can be considered as a sequence modelling problem, as understanding the sequence is important to make any prediction around it.

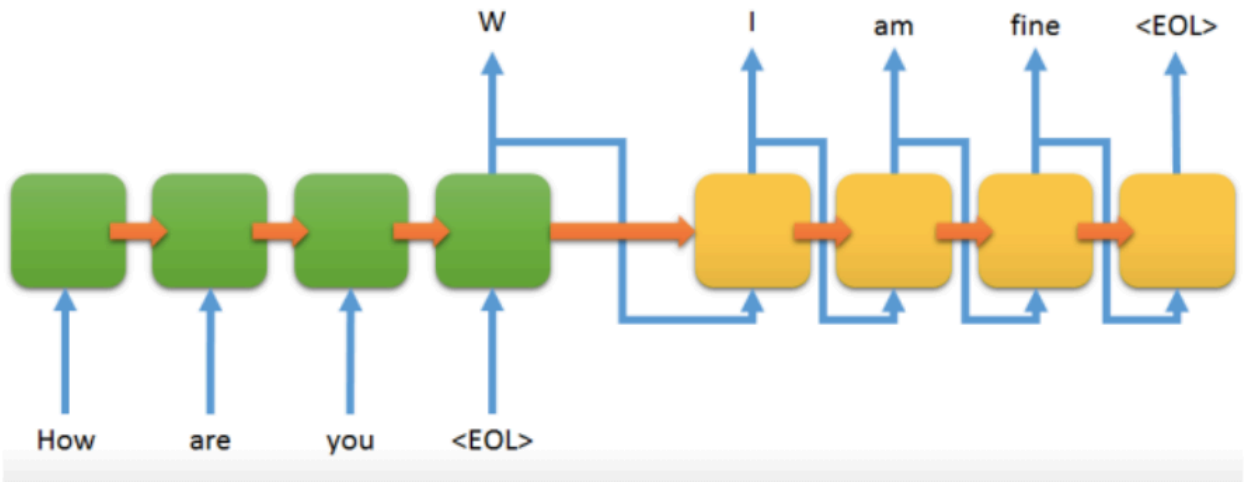


A typical sequence to sequence model has two parts – an **encoder** and a **decoder.** Both the parts are practically two different neural network models combined into one giant network.

Broadly, the task of an encoder network is to understand the input sequence, and create a smaller dimensional representation of it. This representation is then forwarded to a decoder network which generates a sequence of its own that represents the output.



- A RNN layer (or stack thereof) acts as "encoder": it processes the input sequence and returns its own internal state. Note that we discard the outputs of the encoder RNN, only recovering the state. This state will serve as the "context", or "conditioning", of the decoder in the next step.
- Another RNN layer (or stack thereof) acts as "decoder": it is trained to predict the next characters of the target sequence, given previous characters of the target sequence. Specifically, it is trained to turn the target sequences into the same sequences but offset by one timestep in the future, a training process called "teacher forcing" in this context. Importantly, the encoder uses as initial state, the state vectors from the encoder, which is how the decoder obtains information about what it is supposed to generate. Effectively, the decoder learns to generate targets[t+1...] given targets[...t], *conditioned on the input sequence*.

- 

There are four symbols, however, that we need our vocabulary to contain. Seq2seq vocabularies usually reserve the first four spots for these elements:

- **<PAD>**: During training, we'll need to feed our examples to the network in batches. The inputs in these batches all need to be the same width for the network to do its calculation. Our examples, however, are not of the same length. That's why we'll need to pad shorter inputs to bring them to the same width of the batch
- **<EOS>**: This is another necessity of batching as well, but more on the decoder side. It allows us to tell the decoder where a sentence ends, and it allows the decoder to indicate the same thing in its outputs as well.
- **<UNK>**: If you're training your model on real data, you'll find you can vastly improve the resource efficiency of your model by ignoring words that don't show up often enough in your vocabulary to warrant consideration. We replace those with <UNK>.
- **<SOS>**: This is the input to the first time step of the decoder to let the decoder know when to start generating output.

Improving the performance of models – Beam Search and Attention mechanism