# Sequence Models

Dr Anish Roychowdhury

# Motivation for Seq 2 Seq models

*The traditional Deep Neural Networks (DNNs) are powerful machine learning models that achieve excellent performance on difficult problems such as speech recognition and visual object recognition. But they can only be used for large labeled data where inputs and targets can be sensibly encoded with vectors of fixed dimensionality. They cannot be used to map sequences-to-sequences (for eg. machine translation)*

## Why learning about Encoder Decoders is important

The Encoder-Decoder architecture is relatively new and had been adopted as the core technology inside Google’s translate service in late 2016. It forms the basis for advanced sequence-to-sequence models like Attention models, GTP Models, Transformers, and BERT. Hence, understanding it can be very crucial before moving onto advanced mechanisms.

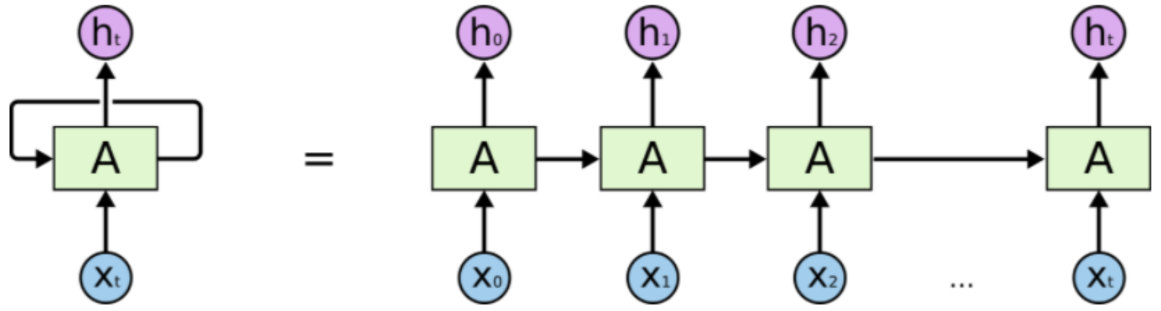
## Movie Reviews : A basic sequence problem

The challenge

Consider a very simple problem of predicting whether a movie review is positive or negative. Here our input is a sequence of words and output is a single number between 0 and 1. If we used traditional DNNs, then we would typically have to encode our input text into a vector of fixed length using techniques like BOW, Word2Vec, etc. But note that here the sequence of words is not preserved and hence when we feed our input vector into the model, it has no idea about the order of words and thus it is missing a very important piece of information about the input

RNNs to the rescue!

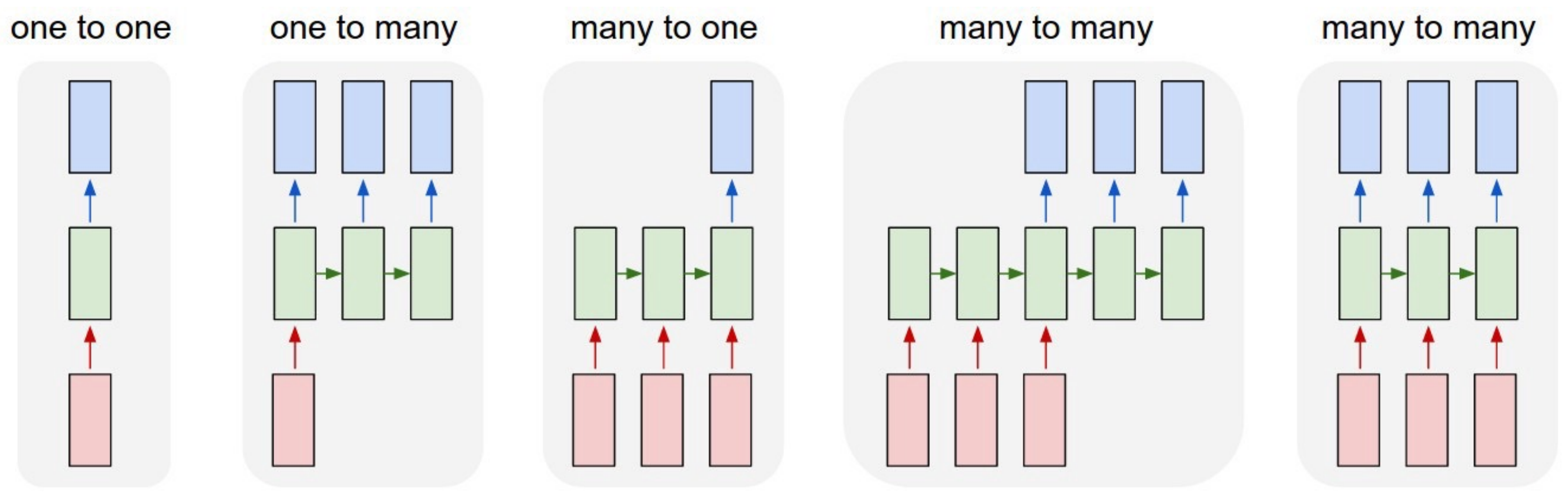
Thus to solve this issue, RNNs came into the picture. In essence, for any input X = ($x_0$, $x_1$, $x_2$, … $x_t$) with a variable number of features, at each time-step, an RNN cell takes an item/token $\mathbf{x_t}$ as input and produces an output $\mathbf{h_t}$ while passing some information onto the next time-step. These outputs can be used according to the problem at hand.



An unrolled recurrent neural network.

# Categories of Seq problems

The movie review prediction problem is an example of a very basic sequence problem called many to one prediction. There are different types of sequence problems for which modified versions of this RNN architecture are used.
Sequence problems can be broadly classified into the following categories:



1 2 3 4 5

Can you think of an application against each ?

# Applications of Seq to Seq Modelling

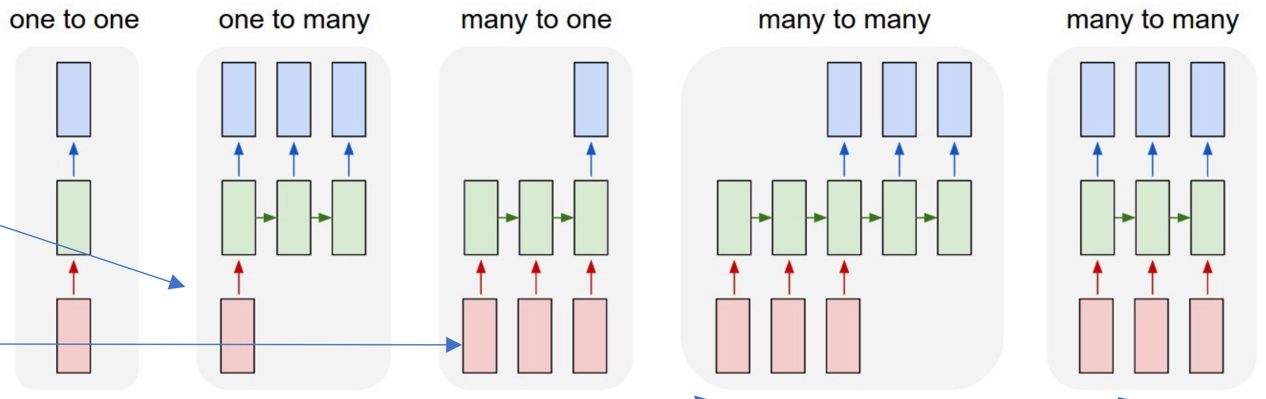
***(1)*** *Vanilla mode of processing without RNN, from fixed-sized input to fixed-sized output (e.g. image classification).*

***(2)*** *Sequence output (e.g. image captioning takes an image and outputs a sentence of words).*

***(3)*** *Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment).*

***(4)*** *Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French).*

***(5)*** *Synced sequence input and output (e.g. video classification where we wish to label each frame of the video).*



# Seq to Seq problems : Machine  Translation

**Sequence-to-Sequence Problems**

Sequence-to-Sequence (Seq2Seq) problems is a special class of Sequence Modelling Problems in which both, the input and the output is a sequence. Encoder-Decoder models were originally built to solve such Seq2Seq problems.

**The Neural Machine Translation Problem**

To help you understand better, I will be taking Neural Machine Translation as a running example throughout this post. In neural machine translation, the input is a series of words, processed one after another. The output is, likewise, a series of words.

**Task**: Predict the French translation for every English sentence used as input.

# Language Translation

## The Language Translation as a sequence Problem

Another important sequence prediction problem is to translate text from one natural language to another. In such a setting, the input sequence is a sentence in the source language, and the predicted output sequence is the corresponding sentence

Why is it tricky ?

In the destination language. It is not necessarily the case that the sentences consist of the same number of words in the two different languages. A good English translation of the French sentence *Je suis étudiant* is

"I am a student," where we see that the English sentence contains one more word than its French counterpart.

For example, translating "What are you doing today?" from English to Chinese has input of 5 words and output of 7 symbols (今天你在做什麼？). Clearly, we can't use a regular LSTM network to map each word from the English sentence to the Chinese sentence.

The Need for Sequence Models

Introduced for the first time in 2014 by Google, a sequence to sequence model aims to map a fixed-length input with a fixed-length output where the length of the input and output may differ.

# Sequence models contd.

## Knowing the Entire sequence is important

Another thing to note is that we want the network to consume the entire input sequence before starting to emit the output sequence, because in many cases, you need to consider the full meaning of a sentence to produce a good translation.

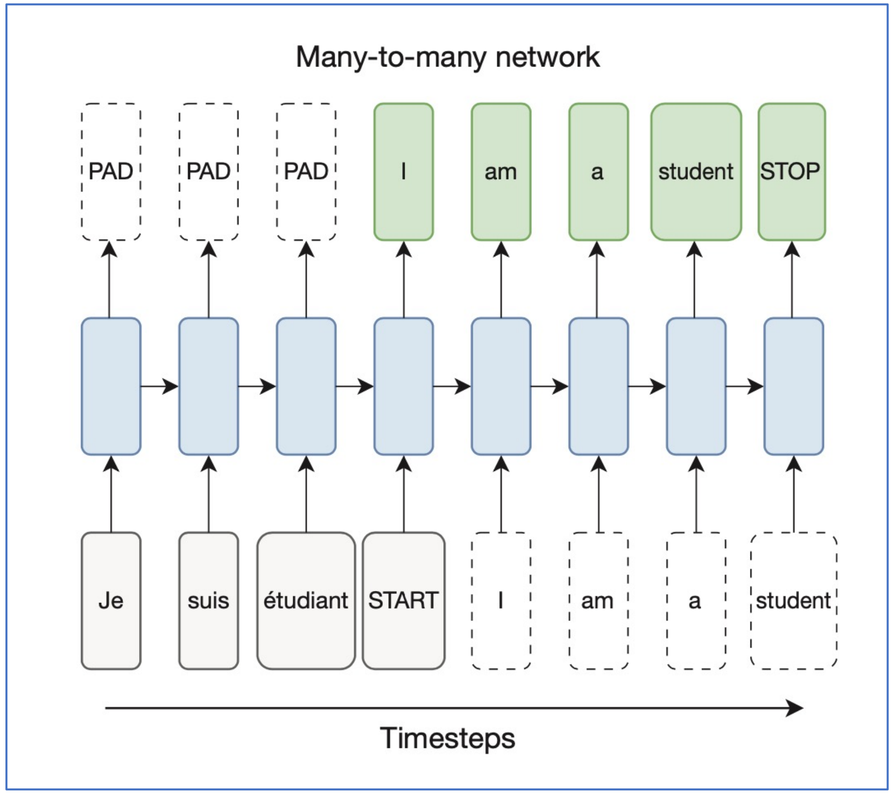
## How do we handle this ?

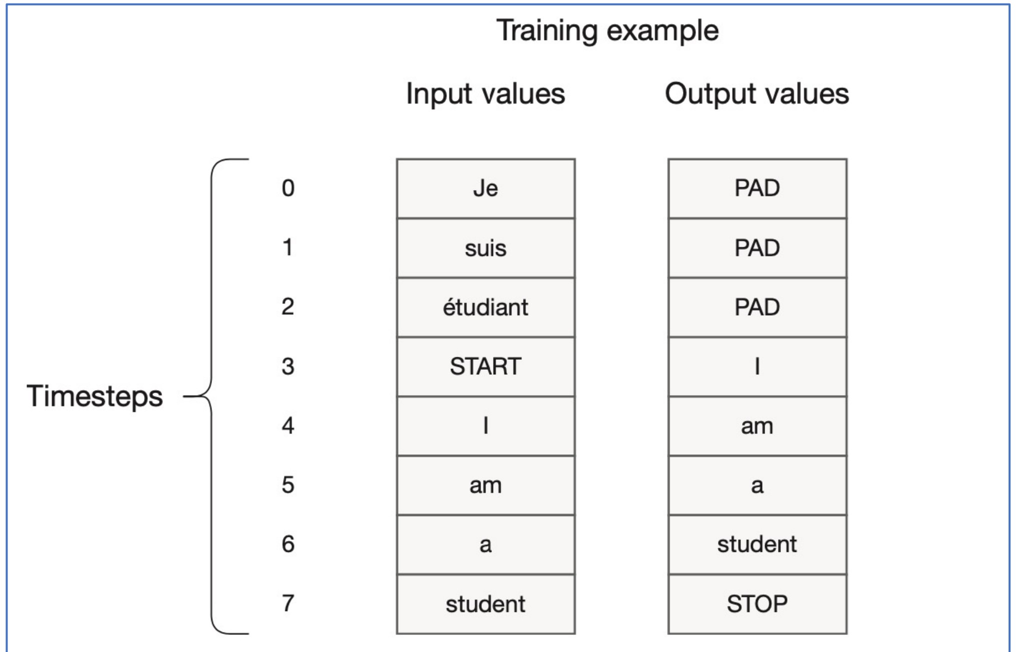
A popular approach to handle this is to teach the network to interpret and emit START and STOP tokens as well as to ignore padding values. Both the padding value and the START and STOP tokens should be values that do not naturally appear in the text.

For example, with words represented by indices that are inputs to an embedding layer, we would simply reserve specific indices for these tokens.

START tokens, STOP tokens, and padding can be used to create training examples that enable many-to-many sequences with variable length

# Sequence models : Language Translation





Training example

| Timesteps | Input values | Output values |
|---|---|---|
| 0 | Je | PAD |
| 1 | suis | PAD |
| 2 | étudiant | PAD |
| 3 | START | I |
| 4 | I | am |
| 5 | am | a |
| 6 | a | student |
| 7 | student | STOP |

*Figure 14-1* Neural machine translation is an example of a many-to-many sequence where the input and output sequences are not necessarily of the same length.

We start with feeding the source sequence to the network, followed by the START token, and then start feeding back its output prediction as input to the next timestep until the network produces a STOP token.
At that point, we have produced the full translated sentence.

# Encoder Decoder Model for Sequence Learning

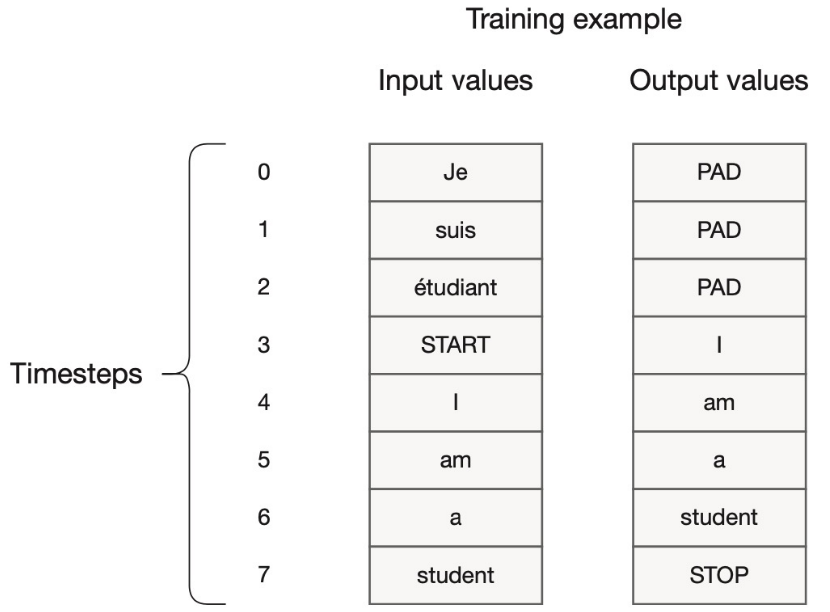
## How does our current model work

Our translation network starts with an accumulated state from seeing the source sequence, is then presented with a single START symbol, and then completes the sentence in the destination language.

That is, during the second half of the translation process, the network simply acts like a neural language model in the destination language.

It turns out that the internal state is all that the network needs to produce the right sentence. we can think of the internal state as a language-independent representation of the overall meaning of the sentence.

Sometimes this internal state is referred to as the *context* or a *thought vector*.



Training example

| Timesteps | Input values | Output values |
|---|---|---|
| 0 | Je | PAD |
| 1 | suis | PAD |
| 2 | étudiant | PAD |
| 3 | START | I |
| 4 | I | am |
| 5 | am | a |
| 6 | a | student |
| 7 | student | STOP |

*Figure 14-1* Neural machine translation is an example of a many-to-many sequence where the input and output sequences are not necessarily of the same length.

Do We see Two different processes involved in the first half and second of the time sequence ?

# Encoder Decoder

## Motivation

Let us consider the first half of the translation process. The goal of this phase is to consume the source sentence and build up this language-independent representation of the meaning of the sentence. Apart from being a somewhat different task than generating a sentence, it is also working with a different language/vocabulary than the second phase of the translation process.

## The question at hand

whether both phases should be handled by the same neural network or if it is better to have two specialized networks.

## The Split

The first network would be specialized in encoding the source sentence into the internal state, and the second network would be specialized in decoding the internal state into a destination sentence. Such an architecture is known as an *encoder-decoder architecture*

# Encoder – Decoder Contd.

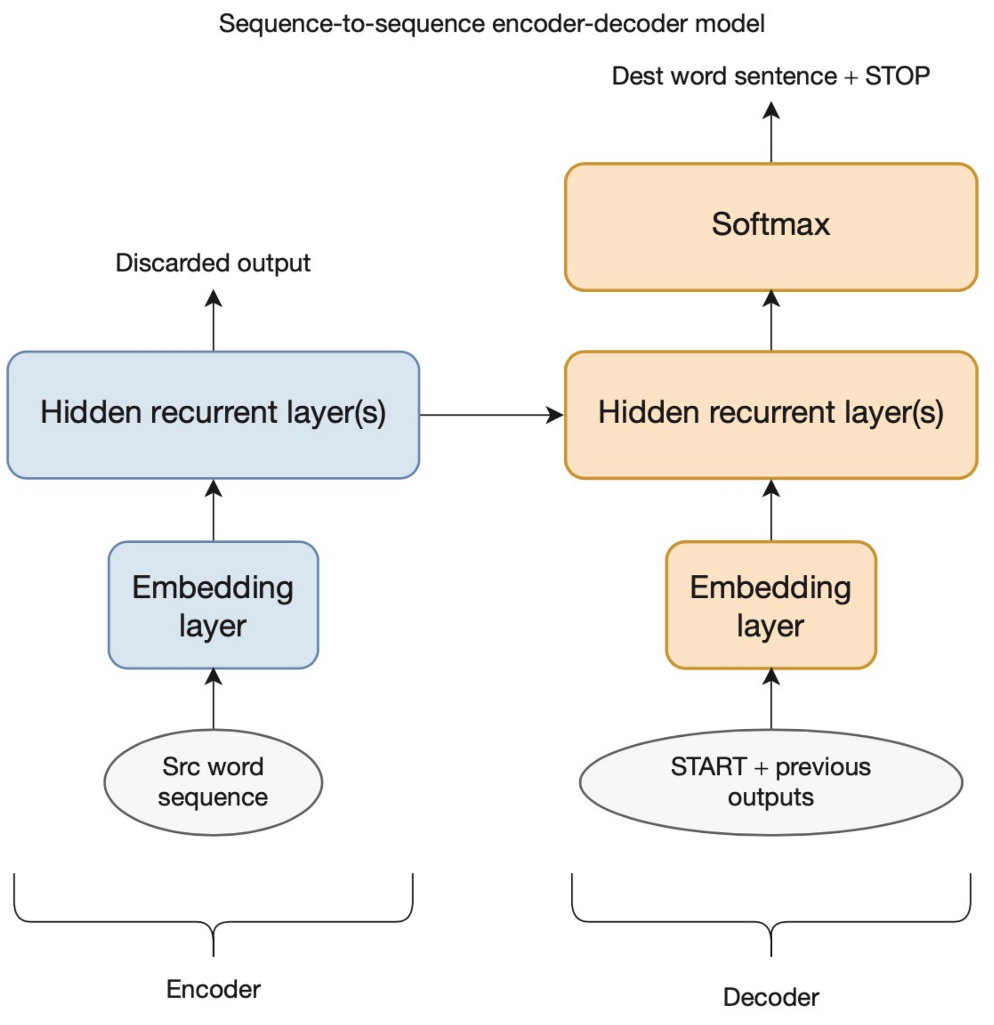

Sequence-to-sequence encoder-decoder model
Dest word sentence + STOP
Softmax
Discarded output
Hidden recurrent layer(s)
Hidden recurrent layer(s)
Embedding layer
Embedding layer
Src word sequence
START + previous outputs
Encoder
Decoder

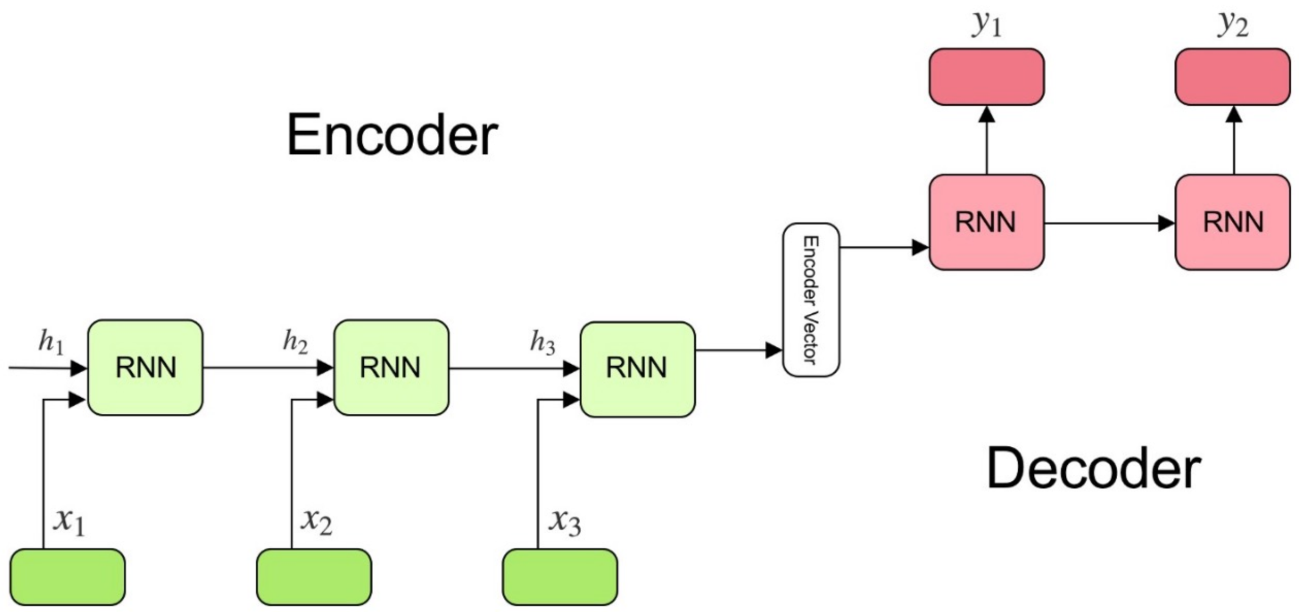
*Figure 14-2* Encoder-decoder model for language translation

In an encoder-decoder architecture, the encoder creates an internal state known as **context** or **thought vector,** which is a **language-independent representation** of the meaning of the sentence.

# Encoder Decoder – the Neural Network view

**How the Sequence to Sequence Model works?**

In order to fully understand the model's underlying logic, we will go over the below illustration:



Encoder-decoder sequence to sequence model

The model consists of 3 parts: encoder, intermediate (encoder) vector and decoder.

# Encoder Decoder by the math!

**Encoder**

- A stack of several recurrent units (LSTM or GRU cells for better performance) where each accepts a single element of the input sequence, collects information for that element and propagates it forward.
- In question-answering problem, the input sequence is a collection of all words from the question. Each word is represented as *x_i* where *i* is the order of that word.
- The hidden states *h_i* are computed using the formula:

$$h_t = f(W^{(hh)} h_{t-1} + W^{(hx)} x_t)$$

This simple formula represents the result of an ordinary recurrent neural network. As you can see, we just apply the appropriate weights to the previous hidden state *h_(t-1)* and the input vector *x_t.*

**Encoder Vector**

- This is the final hidden state produced from the encoder part of the model. It is calculated using the formula above.
- This vector aims to encapsulate the information for all input elements in order to help the decoder make accurate predictions.
- It acts as the initial hidden state of the decoder part of the model.

# Encoder Decoder by the math!

**Decoder**

- A stack of several recurrent units where each predicts an output *y_t* at a time step *t*.
- Each recurrent unit accepts a hidden state from the previous unit and produces and output as well as its own hidden state.
- In the question-answering problem, the output sequence is a collection of all words from the answer. Each word is represented as *y_i* where *i* is the order of that word.
- Any hidden state *h_i* is computed using the formula:

$$h_t = f(W^{(hh)} h_{t-1})$$

- The output *y_t* at time step *t* is computed using the formula:

$$y_t = softmax(W^S h_t)$$

Check out the example here!

Blog Review Time!

https://medium.com/analytics-vidhya/encoder-decoder-seq2seq-models-clearly-explained-c34186fbf49b

# References

## Text book

Magnus Ekman (2022) *Learning Deep Learning: Theory and Practice of Neural Networks, Computer Vision, Natural Language Processing, and Transformers Using TensorFlow* 1st Edition, NVIDIA Deep Learning Institute

## Blogs

https://medium.com/analytics-vidhya/encoder-decoder-seq2seq-models-clearly-explained-c34186fbf49b

https://towardsdatascience.com/understanding-encoder-decoder-sequence-to-sequence-model-679e04af4346

Thank You