Encoders-Decoders, Sequence to Sequence Architecture.

Nadeem · Follow
Published in Analytics Vidhya · 6 min read · Mar 10, 2021

92 2

Understanding Encoders-Decoders, Sequence to Sequence Architecture in Deep Learning.

Translate from one language to another.

In Deep Learning, Many Complex problems can be solved by constructing better neural network architecture. The RNN(Recurrent Neural Network) and its variants are much useful in sequence to sequence learning. The RNN variant LSTM (Long Short-term Memory) is the most used cell in seq-seq learning tasks.

The encoder-decoder architecture for recurrent neural networks is the standard neural ***machine translation method*** *that rivals and in some cases outperforms classical statistical machine translation methods.*

This architecture is very new, having only been pioneered in 2014, although, has been adopted as the core technology inside Google's translate service.

Encoder-Decoder Model

There are three main blocks in the encoder-decoder model,

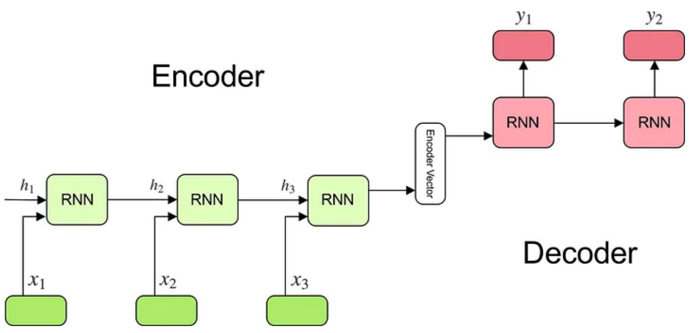
- Encoder
- Hidden Vector
- Decoder

The Encoder will convert the input sequence into a single-dimensional vector (hidden vector). The decoder will convert the hidden vector into the output sequence.

Encoder-Decoder models are jointly trained to maximize the conditional probabilities of the target sequence given the input sequence.

How the Sequence to Sequence Model works?

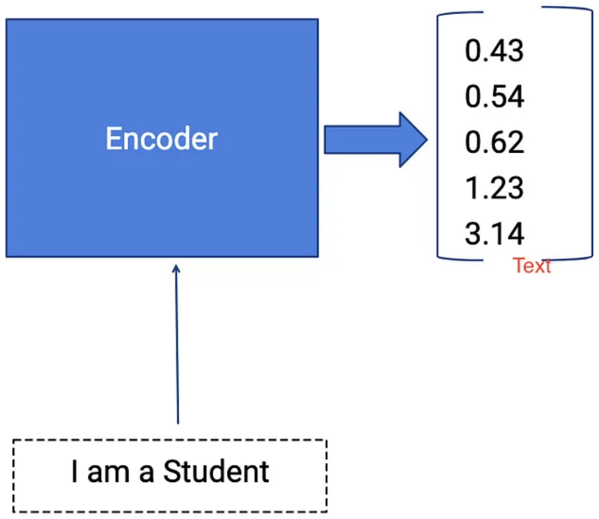
In order to fully understand the model's underlying logic, we will go over the below illustration:



Encoder-decoder sequence to sequence model

Encoder

- Multiple RNN cells can be stacked together to form the encoder. RNN reads each inputs sequentially
- For every timestep (each input) t, the hidden state (hidden vector) h is updated according to the input at that timestep X[i].
- After all the inputs are read by encoder model, the final hidden state of the model represents the context/summary of the whole input sequence.
- Example: Consider the input sequence "I am a Student" to be encoded. There will be totally 4 timesteps (4 tokens) for the Encoder model. At each time step, the hidden state h will be updated using the previous hidden state and the current input.



Example: Encoder

- At the first timestep t1, the previous hidden state h0 will be considered as zero or randomly chosen. So the first RNN cell will update the current hidden state with the first input and h0. Each layer outputs two things — updated hidden state and the output for each stage. The outputs at each stage are rejected and only the hidden states will be propagated to the next layer.
- The hidden states *h_i* are computed using the formula:

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

- At second timestep t2, the hidden state h1 and the second input X[2] will be given as input , and the hidden state h2 will be updated according to both inputs. Then the hidden state h1 will be updated with the new input and will produce the hidden state h2. This happens for all the four stages wrt example taken.
- 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 the 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.

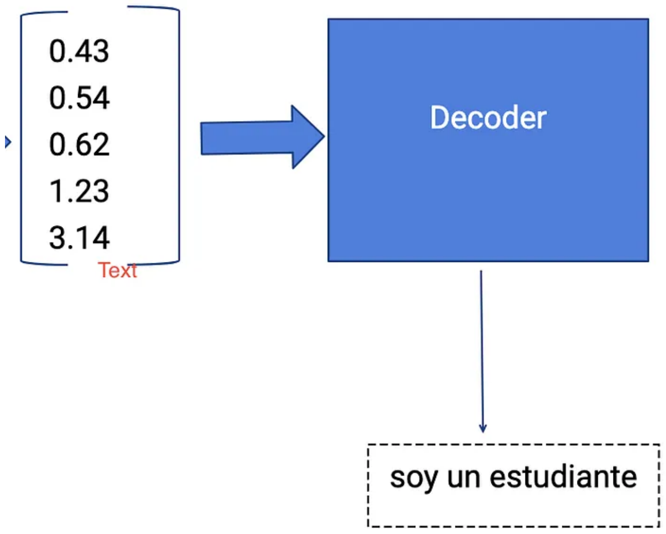
This simple formula represents the result of an ordinary recurrent neural network. As you can see, we just apply the appropriate weights to the previously hidden state h_(t-1) *and the input vector* x_t.

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.

Decoder

- The Decoder generates the output sequence by predicting the next output Yt given the hidden state ht.
- The input for the decoder is the final hidden vector obtained at the end of encoder model.
- Each layer will have three inputs, hidden vector from previous layer ht-1 and the previous layer output yt-1, original hidden vector h.
- At the first layer, the output vector of encoder and the random symbol START, empty hidden state ht-1 will be given as input, the outputs obtained will be y1 and updated hidden state h1 (the information of the output will be subtracted from the hidden vector).
- The second layer will have the updated hidden state h1 and the previous output y1 and original hidden vector h as current inputs, produces the hidden vector h2 and output y2.
- The outputs occurred at each timestep of decoder is the actual output. The model will predict the output until the END symbol occurs.
- 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 an 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.



Example: Decoder.

- Any hidden state *h_i* is computed using the formula:

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

As you can see, we are just using the previous hidden state to compute the next one.

Output Layer

- We use Softmax activation function at the output layer.
- It is used to produce the probability distribution from a vector of values with the target class of high probability.
- The output *y_t* at time step *t* is computed using the formula:

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

We calculate the outputs using the hidden state at the current time step together with the respective weight W(S). Softmax is used to create a probability vector that will help us determine the final output (e.g. word in the question-answering problem).

The power of this model lies in the fact that it can map sequences of different lengths to each other**.** As you can see the inputs and outputs are not correlated and their lengths can differ. This opens a whole new range of problems that can now be solved using such architecture.

Applications

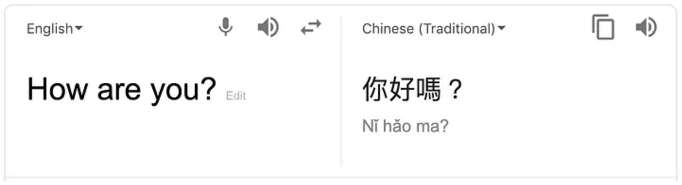
It possesses many applications such as

- Google's Machine Translation
- Question answering chatbots
- Speech recognition
- Time Series Application etc.,

Use Cases of the Sequence to Sequence Model

A sequence to sequence model lies behind numerous systems which you face on a daily basis. For instance, seq2seq model powers applications like Google Translate, voice-enabled devices and online chatbots. Generally speaking, these applications are composed of:

- *Machine translation* — a 2016 paper from Google shows how the seq2seq model's translation quality "approaches or surpasses all currently published results".



How are you?

你好嗎？
Nǐ hǎo ma?

Screenshot of Google translation.

Summary

In this post, you discovered the encoder-decoder model for neural machine translation. How, Encoders and Decoders Model Work

Do you have any questions?
Ask your questions in the comments below and I will do my best to answer.

Further Reading

This section provides more resources on the topic if you are looking to go deeper.

- Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation, 2016.
- Sequence to Sequence Learning with Neural Networks, 2014.
- Presentation for Sequence to Sequence Learning with Neural Networks, 2016.
- Ilya Sutskever Homepage
- Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation, 2014.
- Neural Machine Translation by Jointly Learning to Align and Translate, 2014.
- On the Properties of Neural Machine Translation: Encoder-Decoder Approaches, 2014.
- Kyunghyun Cho Homepage
- Introduction to Neural Machine Translation with GPUs (part1, part2, part3), 2015.