ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DIPARTIMENTO DI
SCIENZE STATISTICHE "PAOLO FORTUNATI"

5 - Sequence to Sequence

Giovanni Della Lunga
giovanni.dellalunga@unibo.it

Halloween Conference in Quantitative Finance

Bologna - October 25-26-27, 2023

Introduction

Introduction

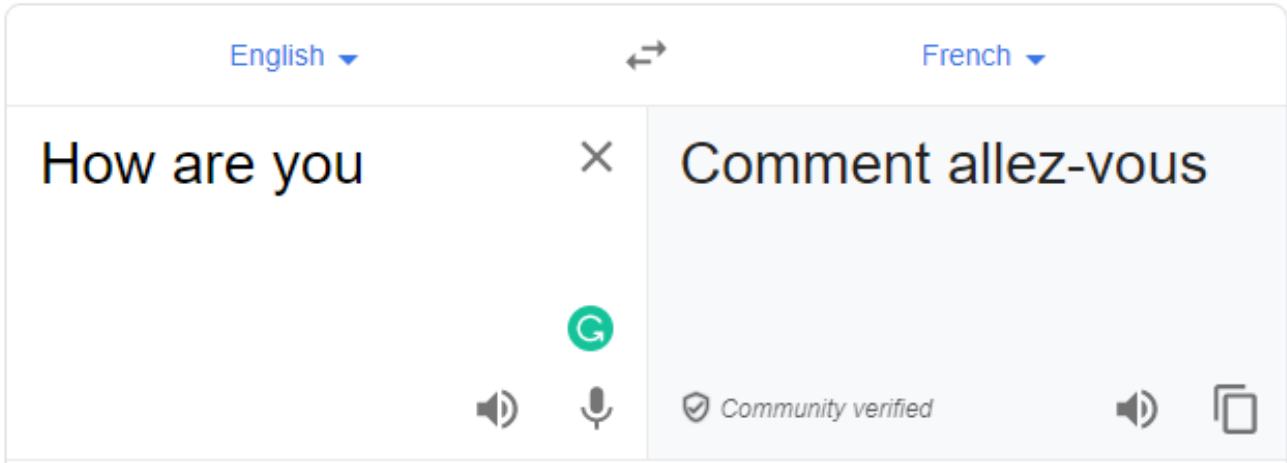
- Sequence to Sequence (often abbreviated to seq2seq) models is a special class of Recurrent Neural Network architectures that we typically use (but not restricted) to solve complex Language problems like Machine Translation, Question Answering, creating Chatbots, Text Summarization, etc.
- This model can be used as a solution to any sequence-based problem, especially ones where **the inputs and outputs have different sizes and categories**.
- Infact RRN e more advanced form of sequence Network (LSTM, GRU) works well when the initial and the final sequence have the same size.

Introduction

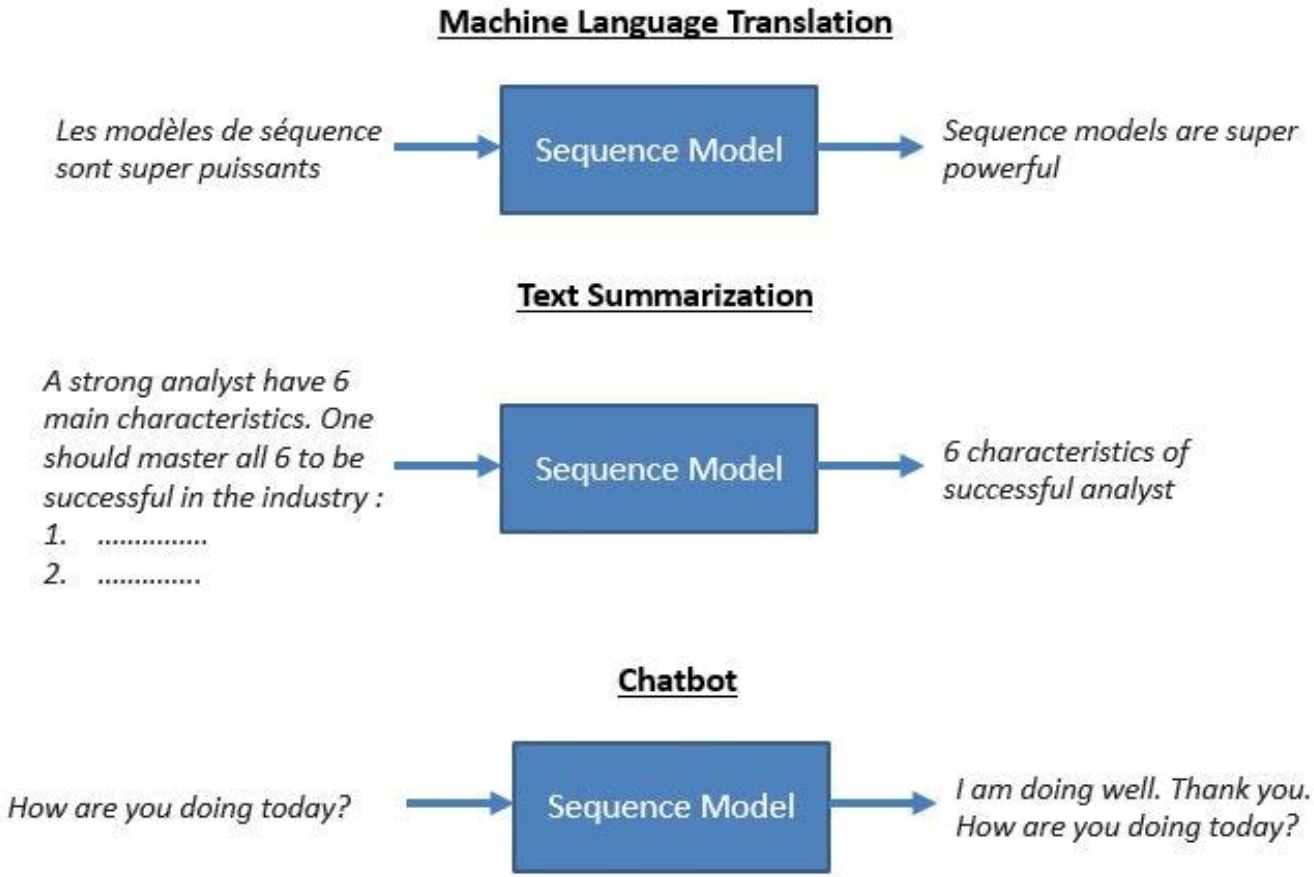
- In the general case, **input sequences and output sequences have different lengths** (e.g. machine translation) and **the entire input sequence is required in order to start predicting the target**.
- This requires a more advanced setup, which is what people commonly refer to when mentioning "sequence to sequence models".

Use Cases of the Sequence to Sequence Models

- Sequence to sequence models lies behind numerous systems that you face on a daily basis.
- For instance, seq2seq model powers applications like Google Translate, voice-enabled devices, and online chatbots.



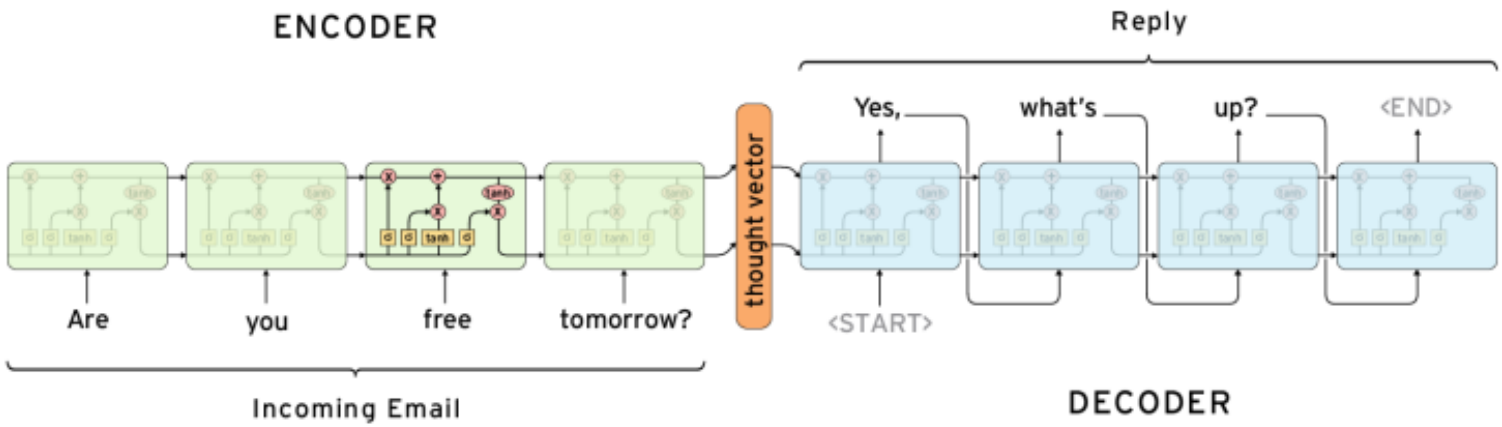
Use Cases of the Sequence to Sequence Models



Encoder-Decoder

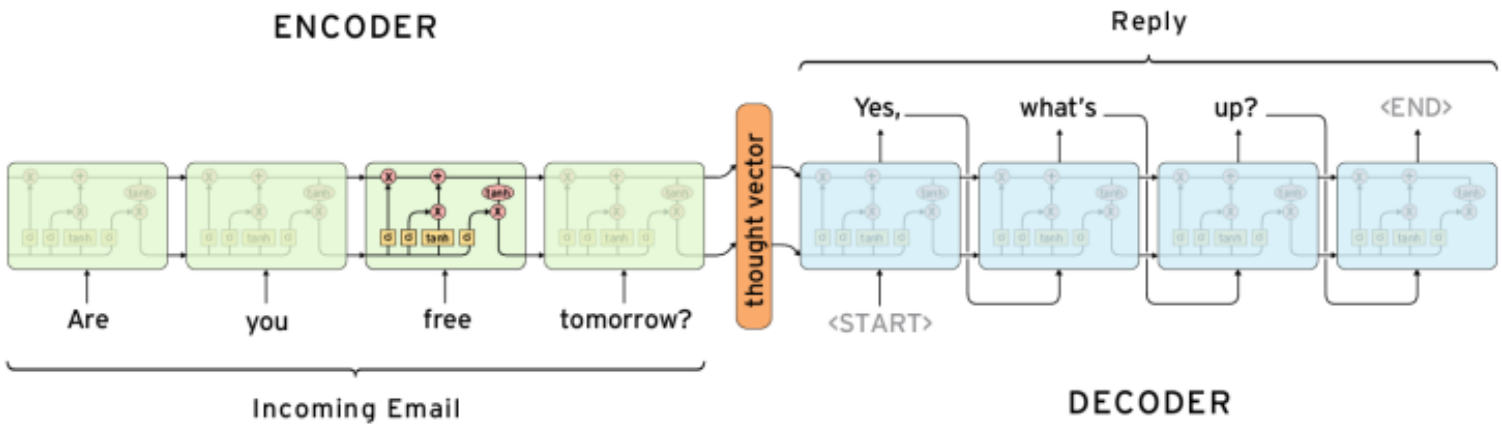
The Encoder-Decoder Architecture

- The encoder-decoder model is a way of organizing recurrent neural networks for sequence-to-sequence prediction problems.
- It was originally developed for machine translation problems, although it has proven successful at related sequence-to-sequence prediction problems such as text summarization and question answering.



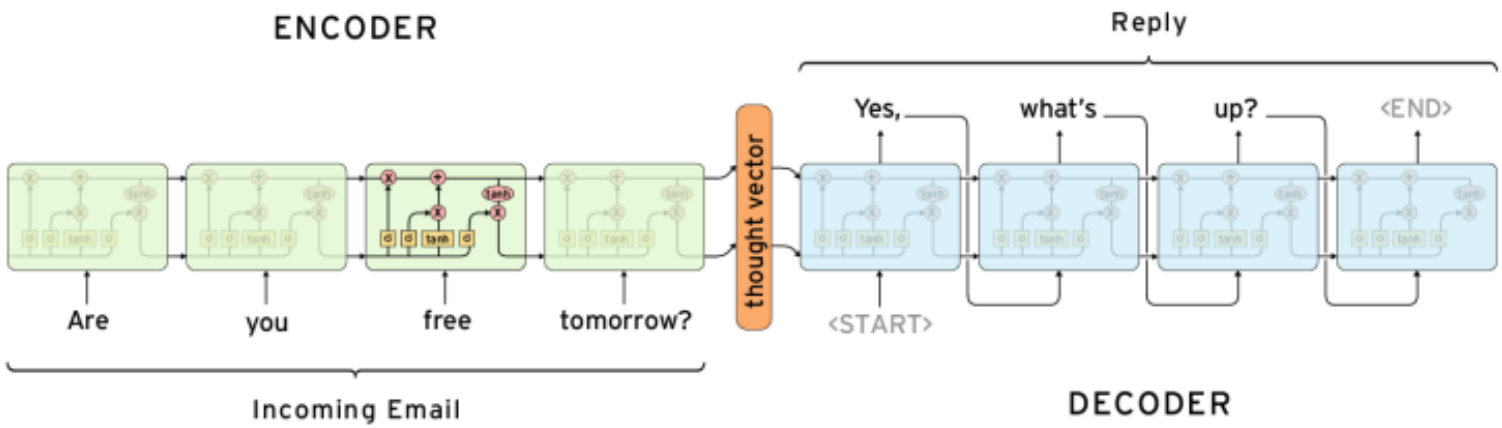
The Encoder-Decoder Architecture

- The approach involves two recurrent neural networks, one to encode the source sequence, called the encoder, and a second to decode the encoded source sequence into the target sequence, called the decoder.
- These components work together to convert sequences of data from one domain into sequences in another.



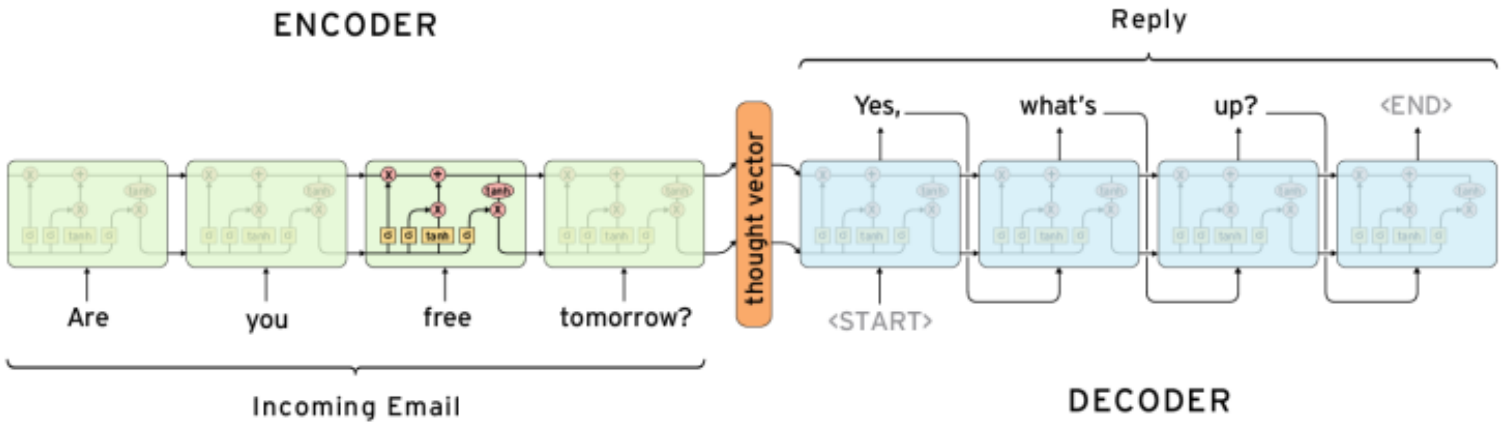
The Encoder-Decoder Architecture

- Both encoder and the decoder are LSTM models (or sometimes GRU models)
- Encoder reads the input sequence and summarizes the information in something called the internal state vectors or context vector (in case of LSTM these are called the hidden state and cell state vectors).
- We discard the outputs of the encoder and only preserve the internal states.



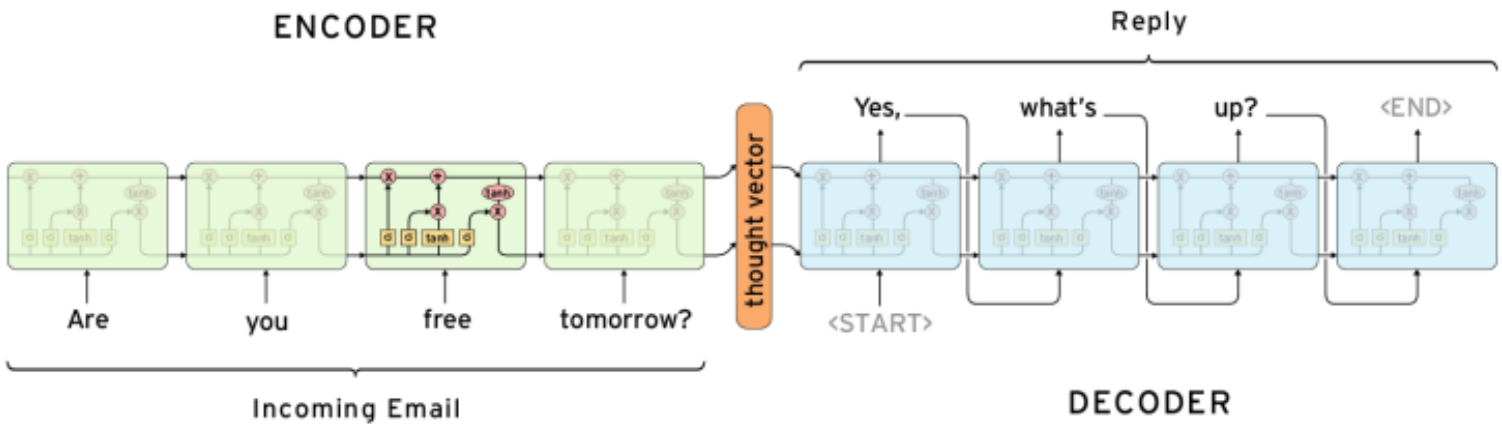
The Encoder-Decoder Architecture

- As the encoder processes the input sequence, it captures the relevant information and context, compressing it into a representation that can be used by the decoder.
- This context vector aims to encapsulate the information for all input elements in order to help the decoder make accurate predictions.



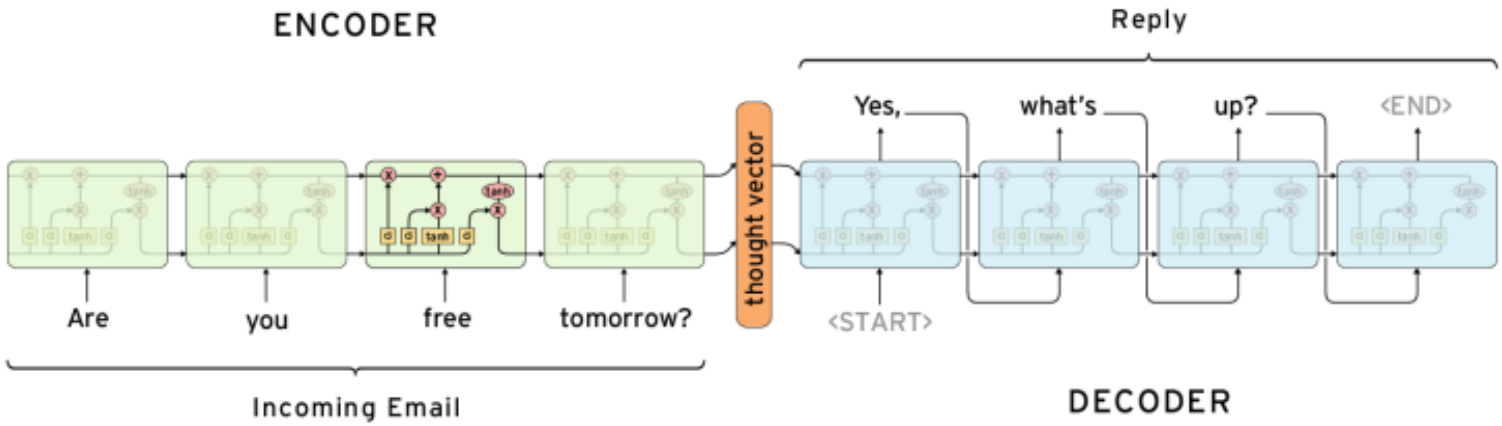
The Encoder-Decoder Architecture

- The decoder is an LSTM whose initial states are initialized to the final states of the Encoder LSTM, i.e. the context vector of the encoder's final cell is input to the first cell of the decoder network.
- Using these initial states, the decoder starts generating the output sequence, and these outputs are also taken into consideration for future outputs.



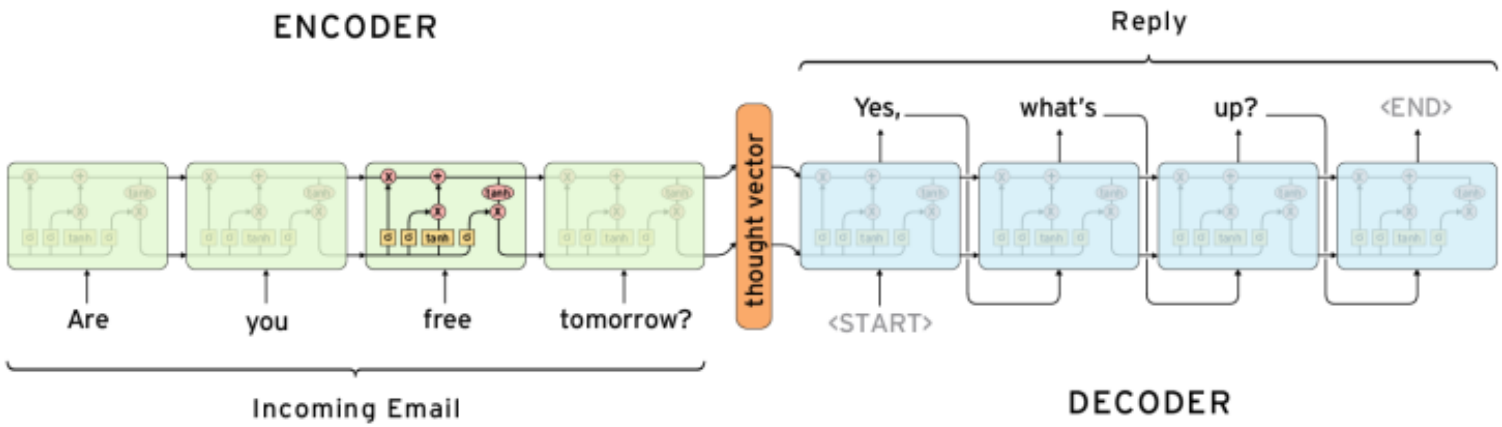
The Encoder-Decoder Architecture

- A stack of several LSTM 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.



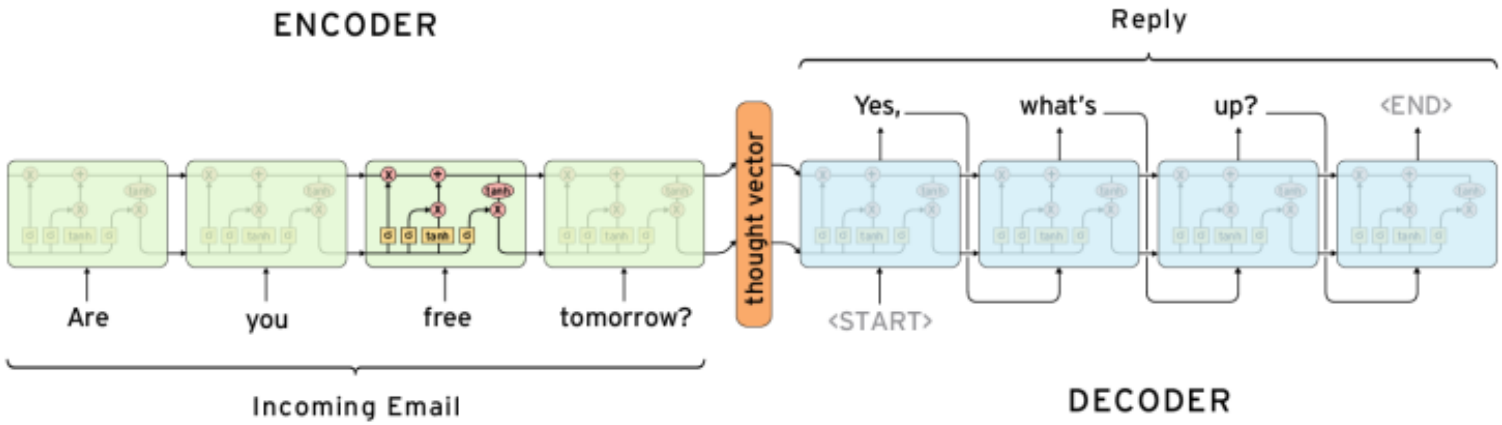
The Encoder-Decoder Architecture

- The most important point is that the initial states of the decoder are set to the final states of the encoder.
- This intuitively means that the decoder is trained to start generating the output sequence depending on the information encoded by the encoder.



The Encoder-Decoder Architecture

- Finally, the loss is calculated on the predicted outputs from each time step and the errors are backpropagated through time in order to update the parameters of the network.
- Training the network over a longer period with a sufficiently large amount of data results in pretty good predictions.



The Encoder-Decoder Architecture

- During inference, we generate one word at a time.
- The initial states of the decoder are set to the final states of the encoder.
- The initial input to the decoder is always the START token.
- At each time step, we preserve the states of the decoder and set them as initial states for the next time step.
- At each time step, the predicted output is fed as input in the next time step.
- We break the loop when the decoder predicts the END token.

Reasons to use Seq2Seq Models

- Seq2Seq models, with their encoder-decoder architecture, have gained prominence due to several compelling reasons:
- Variable-Length Input and Output Sequences:
- Seq2Seq models can handle input and output sequences of variable lengths, making them versatile for tasks where the length of the input and output data can vary greatly.
- This flexibility is especially beneficial in NLP tasks, where sentences or paragraphs can be of different lengths.
- Many real-world problems involve transforming one sequence into another. Seq2Seq models provide a natural solution for these tasks, eliminating the need for handcrafted feature engineering or rule-based systems.

Reasons to use Seq2Seq Models

- Seq2Seq models, with their encoder-decoder architecture, have gained prominence due to several compelling reasons:
- Capturing Contextual Information:
- The encoder-decoder architecture effectively captures contextual information from the input sequence, allowing the model to understand the relationships and dependencies within the data.
- The encoder's hidden states or context vector contain rich representations of the input data, enabling the decoder to generate meaningful and coherent sequences as output.
- This is crucial for tasks like language translation, where the meaning of a word can depend on the words around it.

Reasons to use Seq2Seq Models

- Seq2Seq models, with their encoder-decoder architecture, have gained prominence due to several compelling reasons:
- Transfer Learning:
- Pre-trained Seq2Seq models, such as those based on transformers, have become instrumental in transfer learning.
- Models like BERT, GPT, and T5 have demonstrated remarkable performance across a wide range of NLP tasks.

Limitations of Encoder-Decoder Networks

- **The encoder-decoder neural network works fine for smaller sequences but its performance starts to deteriorate when the size of the input sequence becomes too long**.
- This is because **it becomes difficult for the encoder to compress all the contextual information of a longer sequence into a fixed size vector**.
- Even if the size of the target sequence is smaller, the decoder suffers as a consequence of the long input sequence and the overall predictions could become inaccurate.
- **This is when "attention" comes into the picture**.

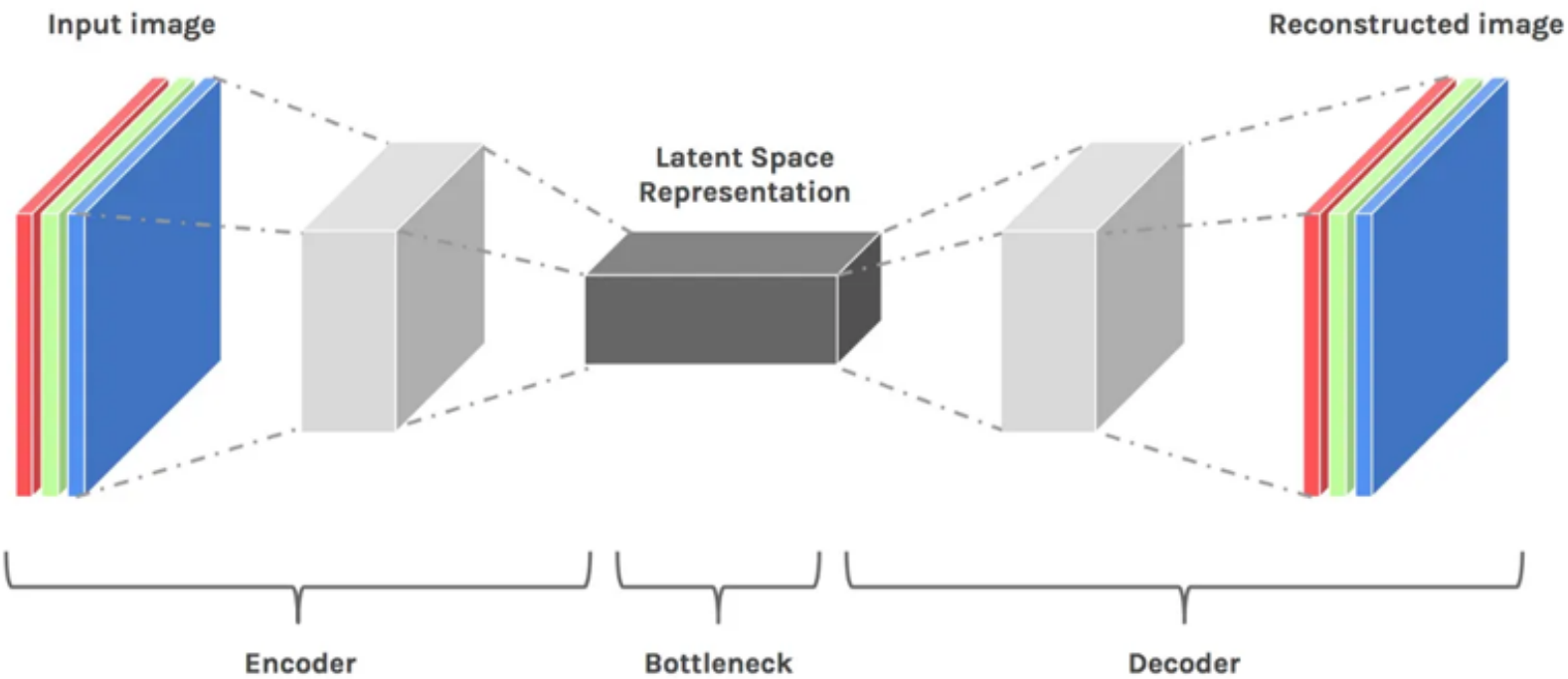
Summary

- In conclusion, sequence-to-sequence models, particularly the encoder-decoder architecture, have revolutionized the way we approach sequence transformation tasks.
- Their ability to handle variable-length input and output sequences, capture contextual information, and provide rich representations makes them indispensable tools in the field of deep learning.
- With ongoing research and advancements, Seq2Seq models continue to drive innovations in NLP, machine translation, summarization, and many other domains, offering transformative solutions to complex sequence-related challenges.
- This architecture is the direct precursor of the modern transformers.

Autoencoders

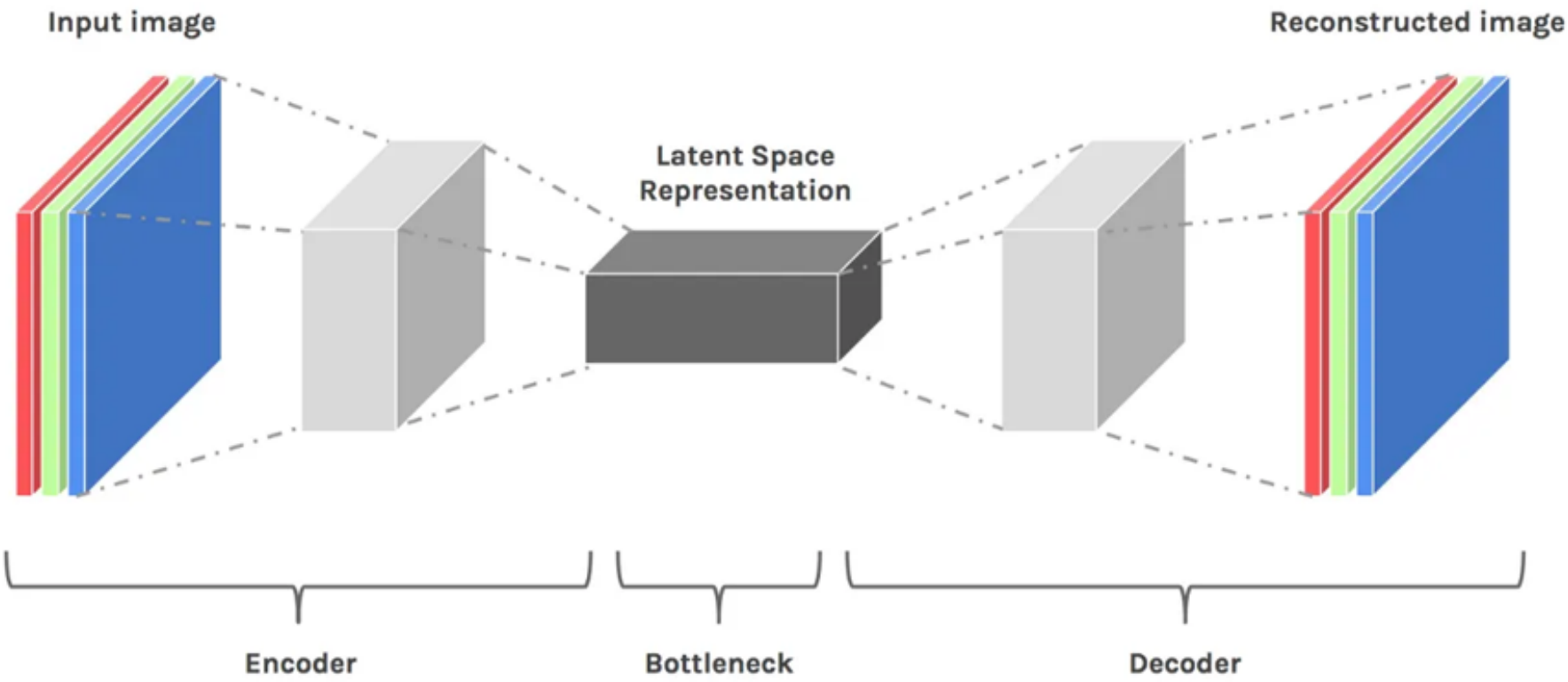
Autoencoders

- Autoencoders are a type of neural network that works in a self-supervised fashion.
- In autoencoders, there are three main building blocks: encoder, decoder, and coder or latent space. And then the decoder does the same but in the opposite order.



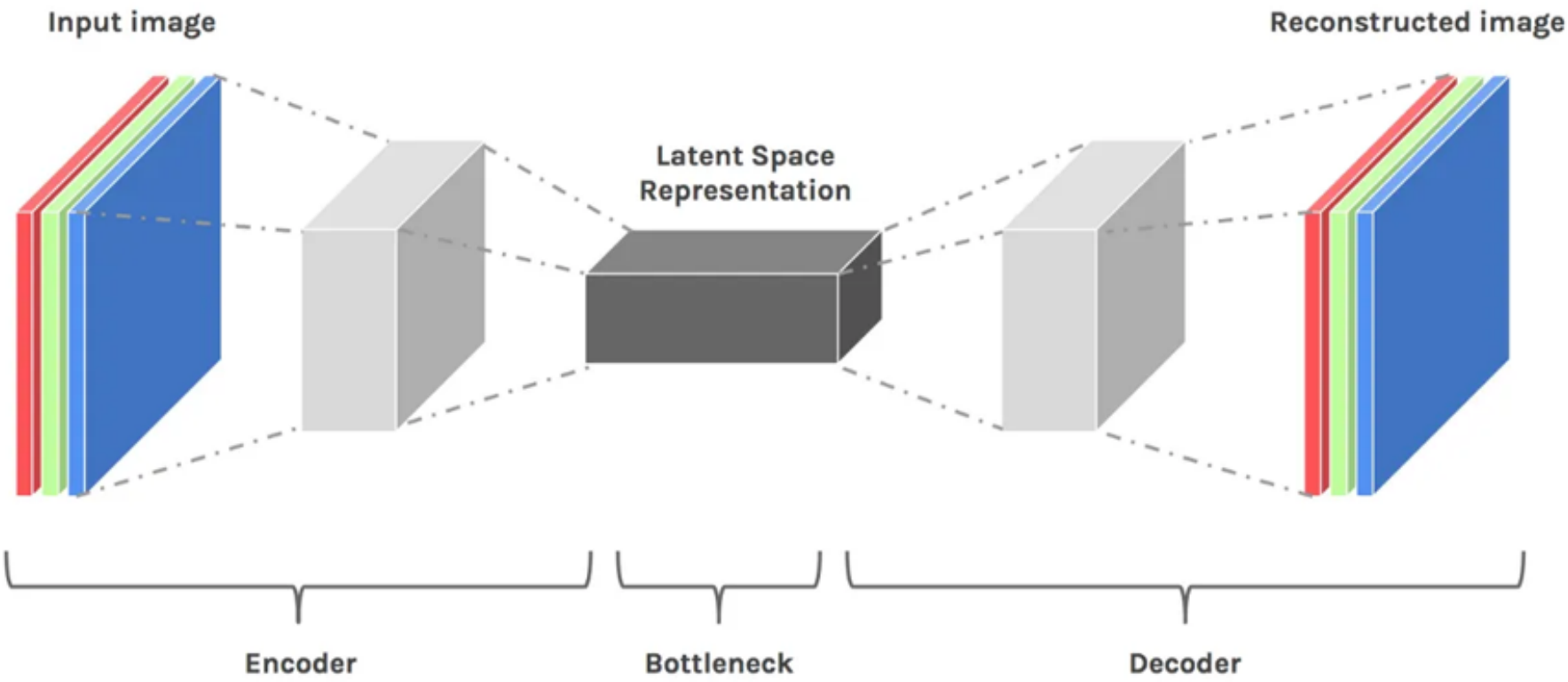
Autoencoders

As usual, you feed the autoencoder with data you want to use and then encoder **encodes or simply extracts useful features of input data and stores it in latent space**. And then the decoder does the same but in the opposite order.



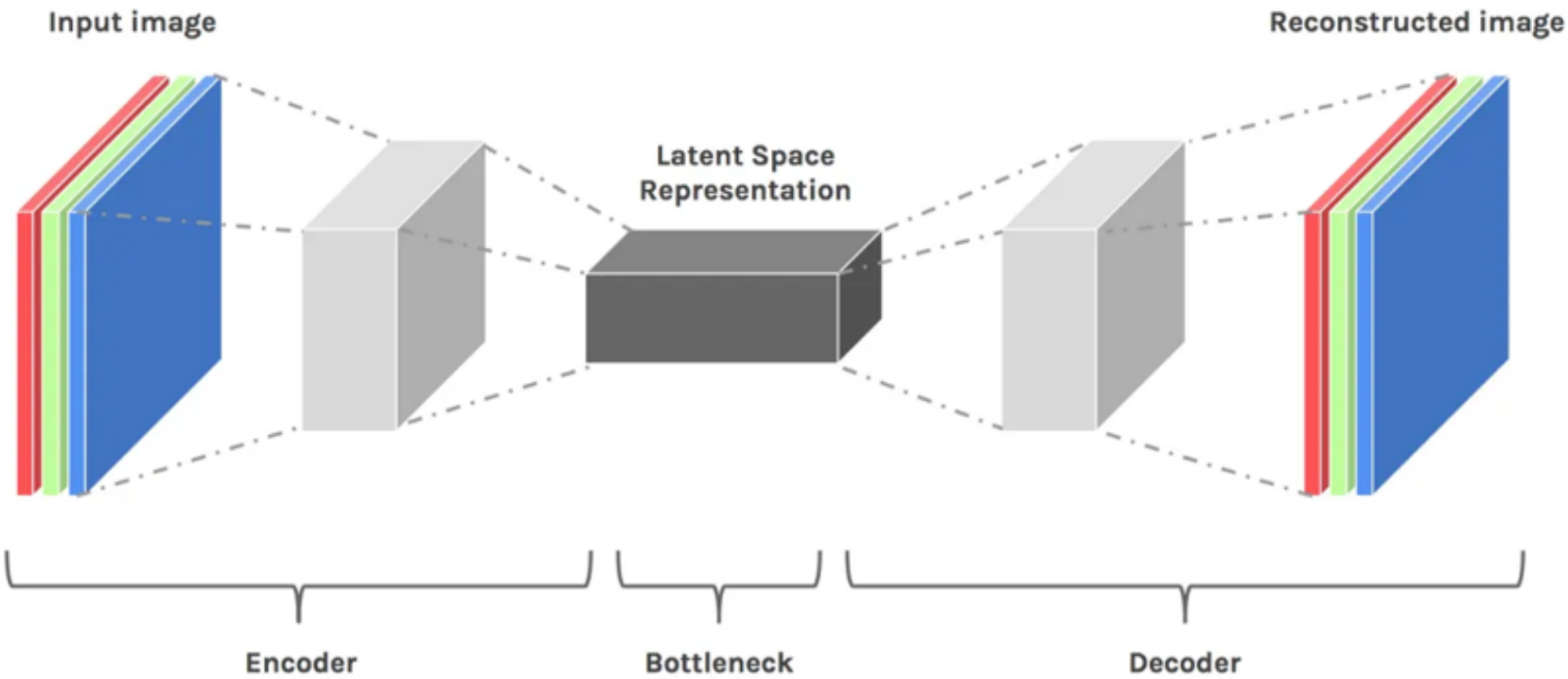
Autoencoders

As usual, you feed the autoencoder with data you want to use and then encoder **encodes or simply extracts useful features of input data and stores it in latent space**. And then the decoder does the same but in the opposite order.



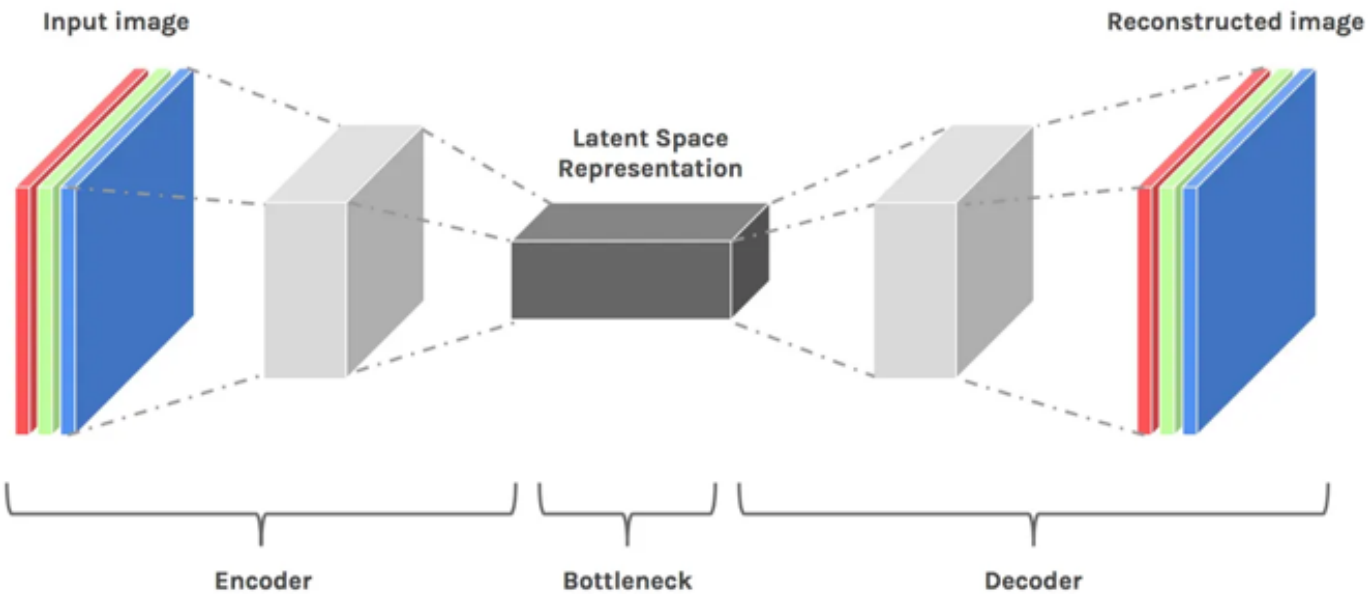
Autoencoders

- Data compression is one of the main advantages of autoencoders, it can be successfully implemented in data transmission problems.
- Imagine server uploads to internet only **Latent Space Representation** of the input image, depending on your request.



Autoencoders

- The main task of undercomplete autoencoders are not just copying input to the output, but instead, learn useful properties of input data.
- Autoencoder whose latent representation of input data dimension is less than the input dimension is called undercomplete.
- This type of autoencoder enables us **to capture the most notable features of the training data**.



Autoencoders

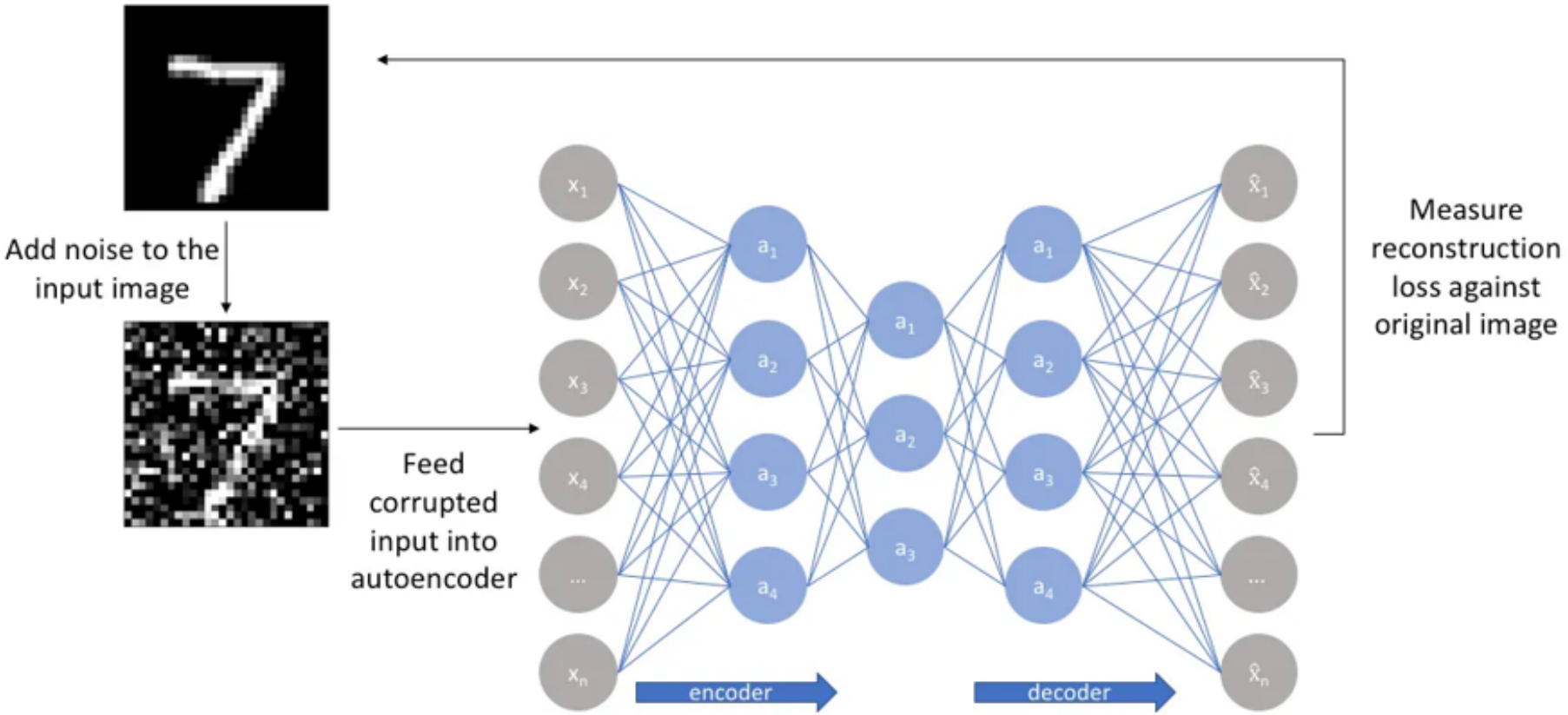
- **Data Reduction**
- By admitting that neural networks can learn nonlinear relationships in data, undercomplete autoencoders become a more powerful and nonlinear form of PCA for higher-dimensional data;
- Autoencoders are capable of learning complex representation of data, which can be then used in visualization in low-dimension space.

Autoencoders

- **Denoising Data**
- Quality loss of data (images/audio/video) is one main problem/issue in data transmission. Due to connection status or bandwidth, data such as image and audio can lose in quality, therefore the problem of denoising data arises.
- Denoising data is one of the cool advantages of autoencoders,
- Ideally, we would like to have our autoencoder is sensitive to reconstruct our latent space representation and insensitive enough to generalize the encoding of input data.
- One of the ways of such a method is to corrupt the input data, adding random noise to input data and then encode that corrupted data.

Autoencoders

Denoising Data

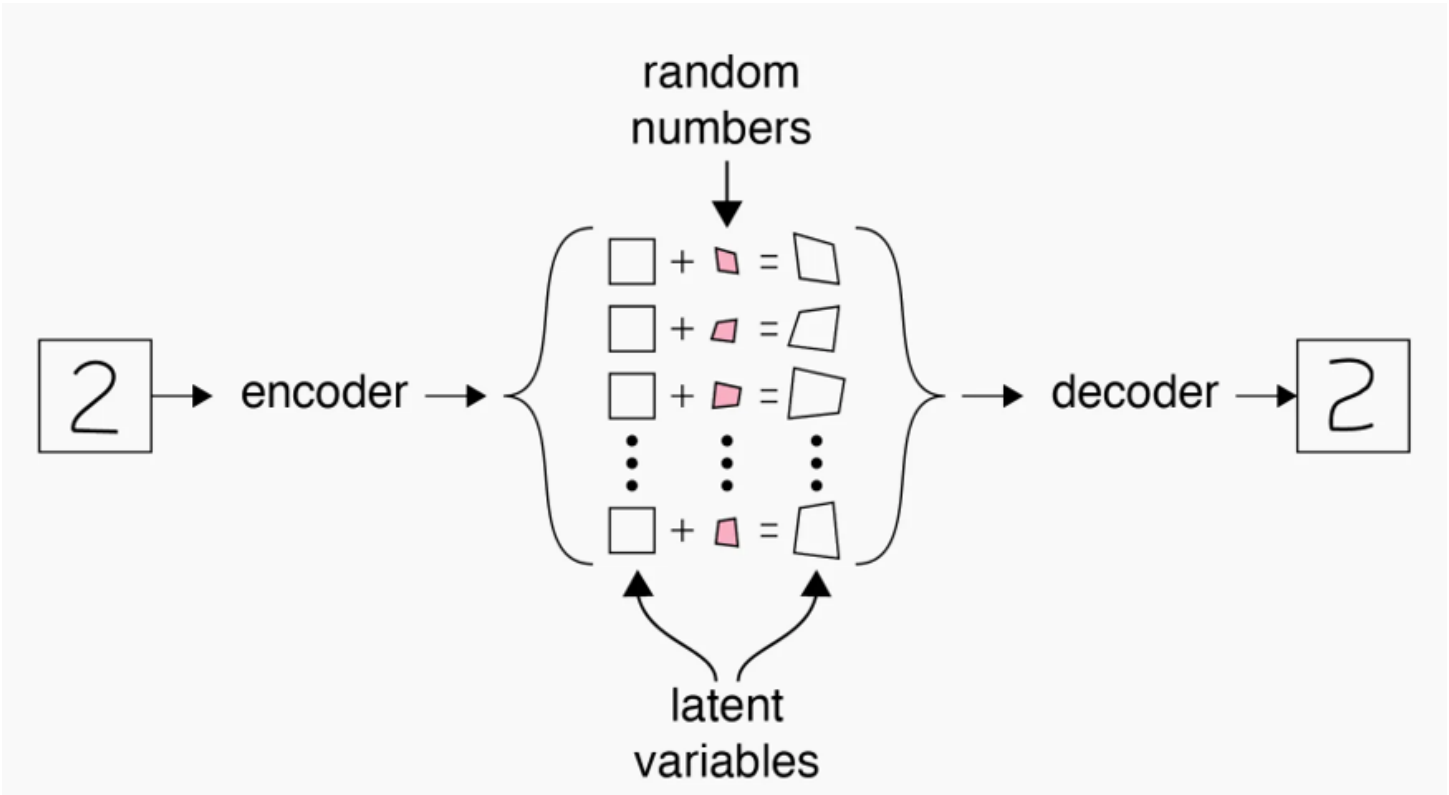


Variational Autoencoders

- VAEs inherit the architecture of traditional autoencoders and use this to learn a data generating distribution, which allows us to take random samples from the latent space.
- These random samples can then be decoded using the decoder network to generate unique data that have similar characteristics to those that the network was trained on.

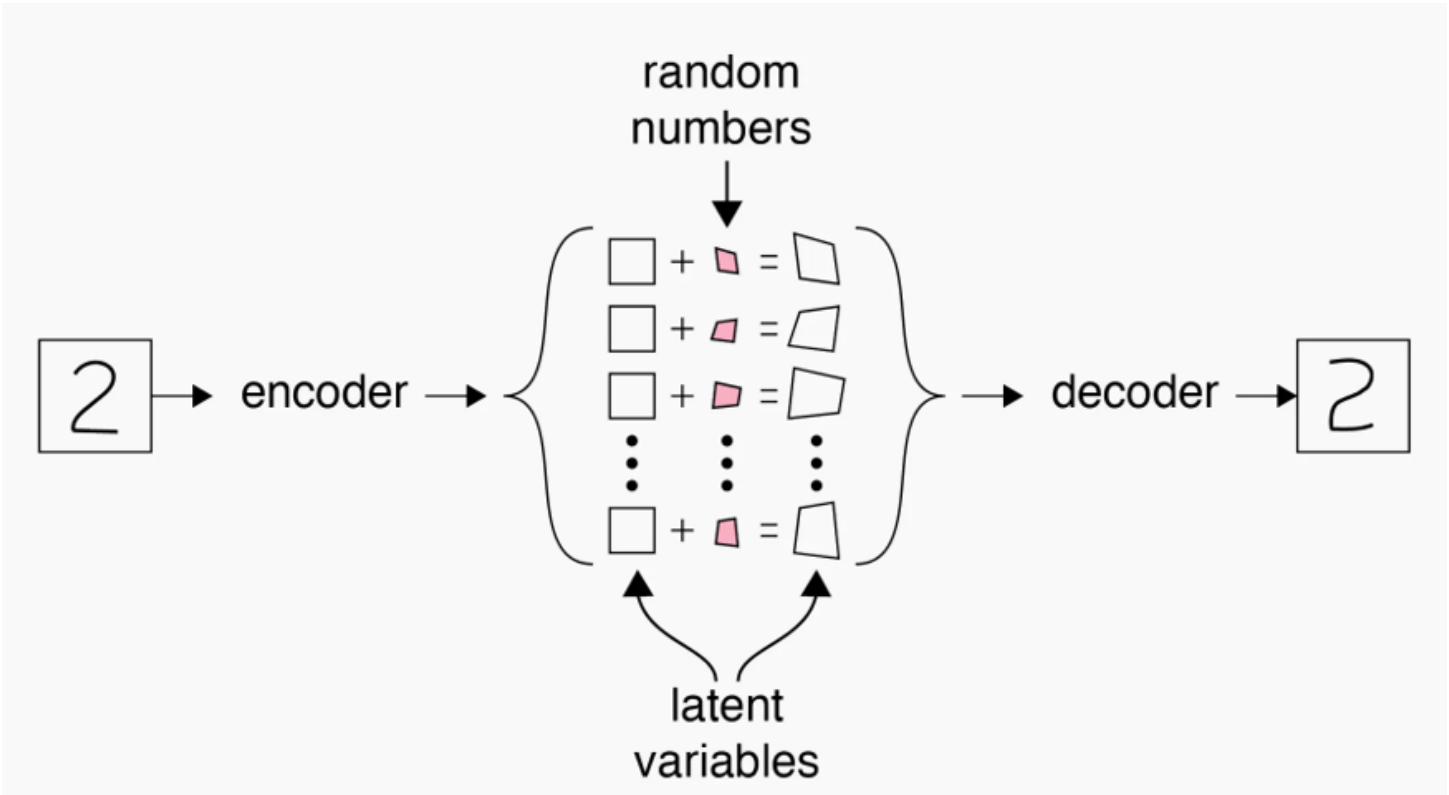
Variational Autoencoders

- **The Data Generating Procedure**
- Looking at the below image, we can consider that our approximation to the data generating procedure decides that we want to generate the number ‘2‘, so it generates the value 2 from the latent variable centroid.



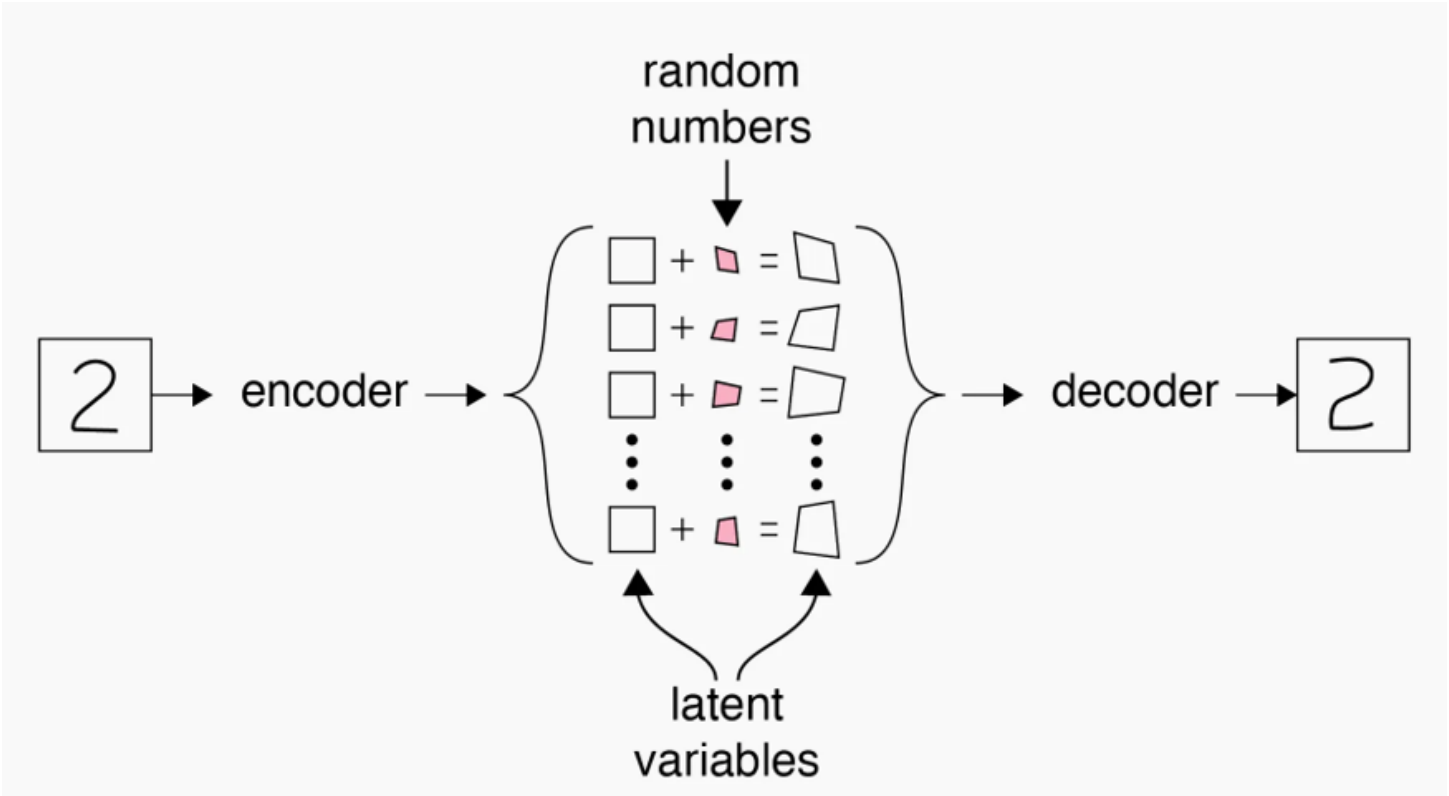
Variational Autoencoders

- **The Data Generating Procedure**
- However, we may not want to generate the same looking ‘2‘ every time so we add some random noise to this item in the latent space, which is based on a random number and the ‘learned‘ spread of the distribution for the value ‘2‘.



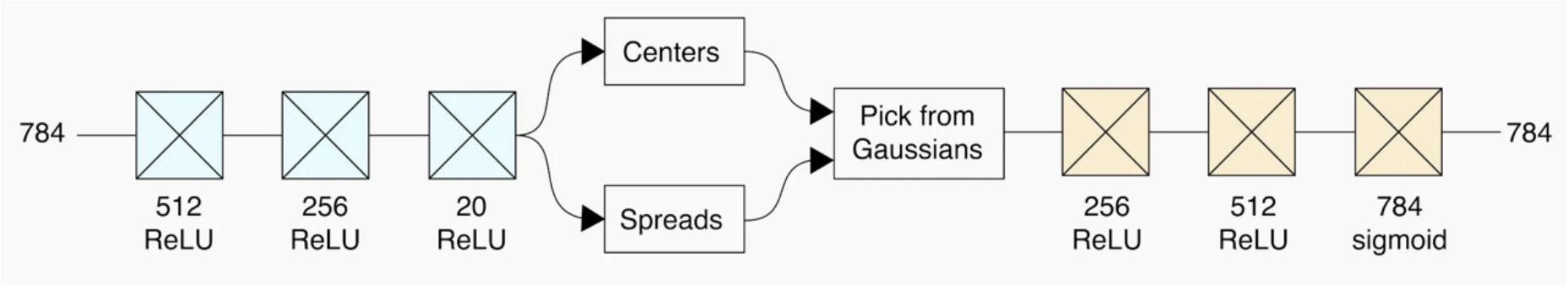
Variational Autoencoders

- **The Data Generating Procedure**
- We pass this through our decoder network and we get a 2 which looks different to the original.



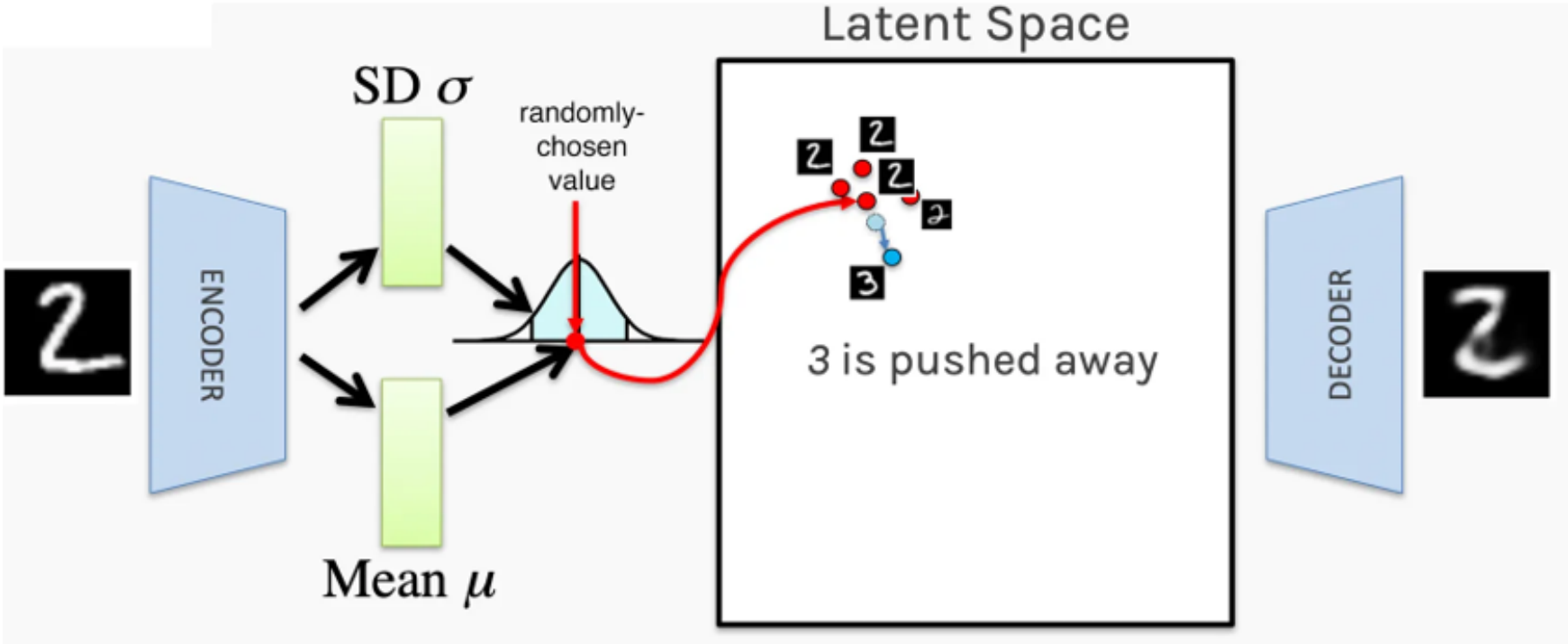
Variational Autoencoders

- The previous one was an oversimplified version which abstracted the architecture of the actual autoencoder network.
- Below is a representation of the architecture of a real variational autoencoder using convolutional layers in the encoder and decoder networks.
- We see that we are learning the centers and spreads of the data generating distributions within the latent space separately, and then ‘sampling‘ from these distributions to generate essentially ‘fake‘ data.



Variational Autoencoders

The inherent nature of the learning procedure means that parameters that look similar (stimulate the same network neurons to fire) are clustered together in the latent space, and are not spaced arbitrarily. This is illustrated in the figure below. We see that our values of 2's begin to cluster together, whilst the value 3 gradually becomes pushed away. This is useful as it means the network does not arbitrarily place characters in the latent space, making the transitions between values less spurious.



Variational Autoencoders

We train the autoencoder using a set of images to learn our mean and standard deviations within the latent space, which forms our data generating distribution. Next, when we want to generate a similar image, we sample from one of the centroids within the latent space, distort it slightly using our standard deviation and some random error, and then pass this through the decoder network. It is clear from this example that the final output looks similar, but not the same, as the input image.