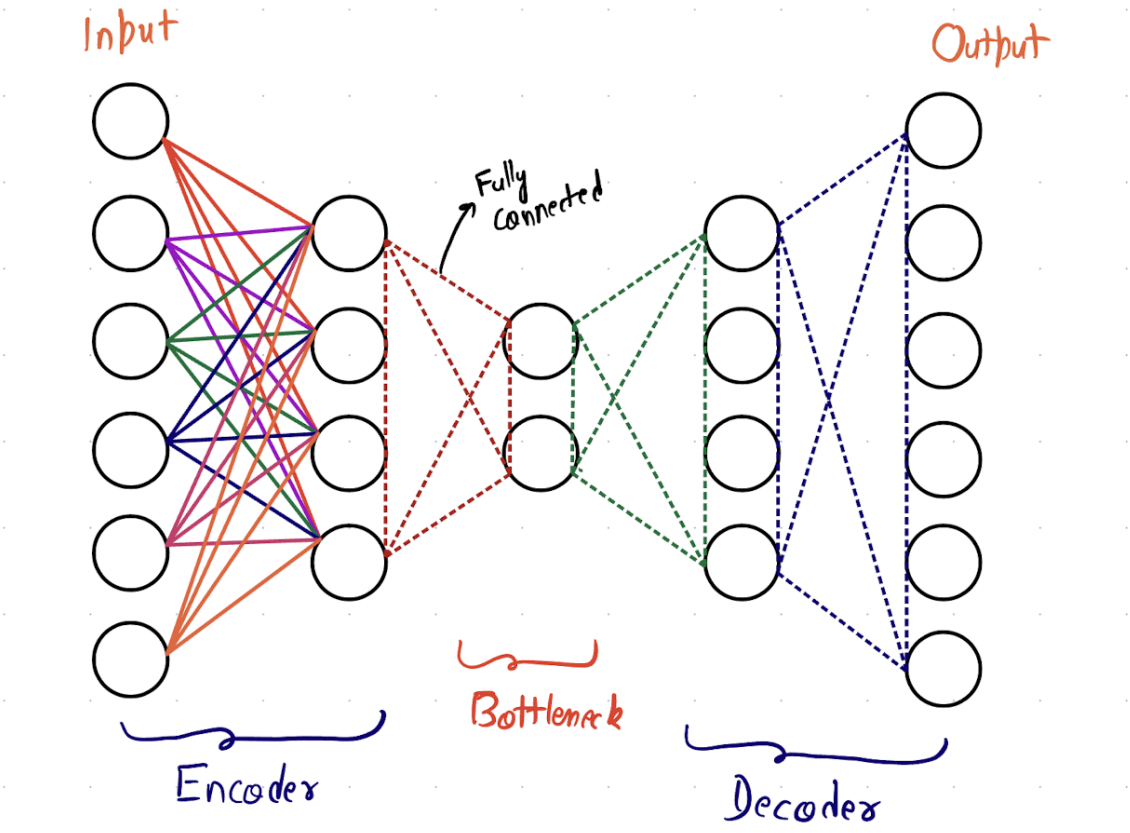
Autoencoders

What are autoencoders (AE)?

- Autoencoders are self supervised learning method where the input (x_i) is same as output ($\hat{x}_i$)
- The network comprises of 3 parts:
  - **Encoder**
  - **Bottleneck layer** => used as encoding/embedding
  - **Decoder**



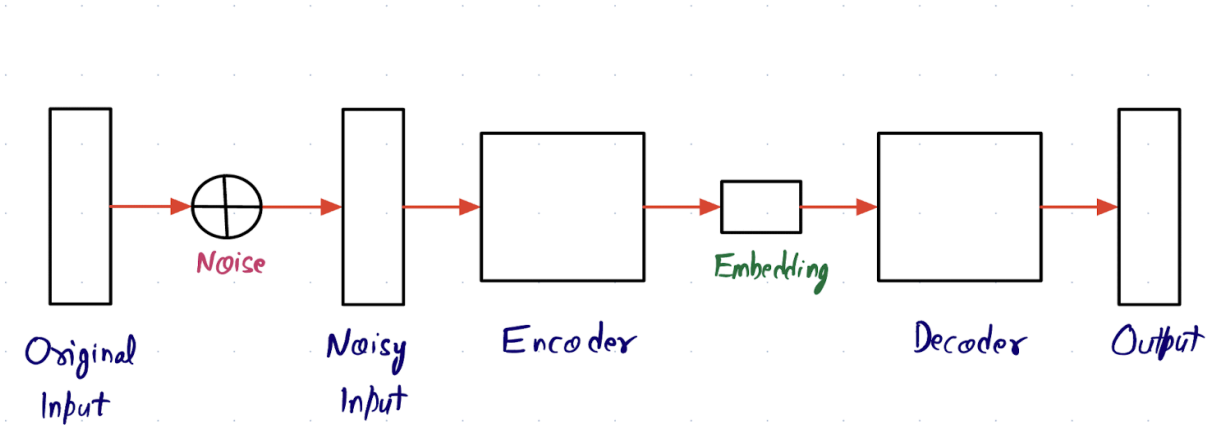
  - The encoder part compresses the input to lower dimensionality embedding/encoding
  - The decoder produces output by expanding this lower dimensionality embedding and tries to reconstruct input from it.
  - **Goal:** $x_i \sim \hat{x}_i$

Do note: It is also called unsupervised learning as they don't need labels to train on.

Applications

- **Dimensionality Reduction**
  - We can use AE to reduce the dimensionality of the data.
  - Do note that
    - The dim. Reduction is data-specific
      - For example: If the AE has been trained on handwritten digits, we can't expect it to compress cats and dogs images.
      - It'll be able to meaningfully compress data similar to what it has been trained on
    - The output of the decoder will not be exactly the same as input i.e. there'll be loss of information.
  - **Code:** Link

- **Denoising AE**
  - In order to make sure that AE doesn't overfit i.e. it doesn't simply learn to copy input to output
    - We add random noise to that data



What happens when we add random noise?

- If AE recreates the noisy input, it means it has overfitted
- Think of it as regularizing the data
  - As there is no pattern to noise, network shouldn't recreate it
- **Code:** Link

- **Recommender Sys using AE**
    - We can use AE to generate embeddings to find similar items (i.e. as a Recommender System)
    - In order to do so
        - We feed sparse data as input to the network
        - Learn the dense embeddings
        - Find similar items using dense embeddings
    - For example:
        - Find similar movies for a given user-item interaction matrix
            - We feed movie vector (item vector) as input to AE
            - The network learns the dense embeddings.
            - Using cosine similarity on these embeddings, we find similar movies (the higher the score, the more similar the movie).
    - **Code: Link**

Is it necessary for the encoder and decoder to be symmetric?

- Not necessarily.
- Earlier we used to keep them symmetric i.e. k^{th} and $n - k^{th}$ layer will have the same number of neurons

  Why did we keep the network symmetric?
    - It was because of weight sharing (weight tying)
    - Weights were shared between the encoder and decoder
    - To reduce the number of parameters.