NLP: Day 3

Recurrent Neural Network (RNN):

- **Artificial Neural Network (ANN):**
  - General-purpose neural network architecture.
  - Suitable for a wide range of data types and problem domains.
  - Can be used for tabular data, where each row represents a sample and each column represents a feature.
  - Example: Predicting house prices based on features like area, number of bedrooms, and location.
  - Can also be used for image classification by flattening the image into a vector.
  - Example: Identifying handwritten digits by treating each pixel as a feature.
- **Convolutional Neural Network (CNN):**
  - Specifically designed for processing grid-like data, such as images.
  - Utilizes convolutional layers to extract local patterns and hierarchies of features.
  - Best suited for image and video-related tasks.
  - Ideal for 2D or 3D data like images or volumes.
  - Each pixel or voxel is considered a feature.
  - Example: Classifying objects in a 32x32x3 image (width x height x number of colour channels).
- **Recurrent Neural Network (RNN):**
  - Specialized for processing sequential data that has temporal dependencies.
  - Suitable for data that has an ordered sequence of elements, such as text, time series, speech, or DNA sequences.
  - Processes data in a step-by-step manner, maintaining internal memory.
  - Each step considers the current input and the previous step's output.
  - Example: Language modelling, where the model predicts the next word in a sentence based on the context of the previous words.
  - Example of RNN for text generation:
    - Input sequence: "The cat sat"
    - RNN processes "The" and generates "cat"
    - RNN processes "cat" and generates "sat"
    - RNN processes "sat" and generates the end-of-sequence token or predicts the next word based on the context.
  - Example of RNN for time series prediction:
    - Input sequence: [10, 15, 20, 25]
    - RNN processes 10 and predicts 15
    - RNN processes 15 and predicts 20
    - RNN processes 20 and predicts 25
    - RNN continues to predict the next value in the time series.

Q. Why use RNN?

- RNN (Recurrent Neural Network) is specifically designed for processing sequential data.
- It can capture and utilize the sequential dependencies and temporal information present in the data.
- RNN is suitable for tasks where the order of data elements matters, such as natural language processing, speech recognition, and time series analysis.
- RNN maintains an internal memory that allows it to process inputs in a step-by-step manner while retaining information from previous steps.

Explanation with an example:

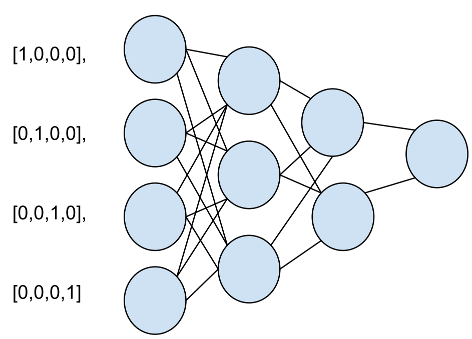
Input	Output
My name is Subhash	0
Data Scientist	1
I love AI	0

One Hot Encoding: 1st find the biggest dimension in an array

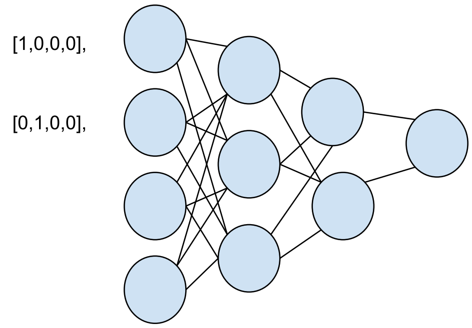
- **Row 1:** [[1,0,0,0], [0,1,0,0], [0,0,1,0], [0,0,0,1]]
- **Row 2:** [[1,0,0,0], [0,1,0,0]]
- **Row 2:** [[1,0,0,0], [0,1,0,0], [0,0,1,0]]

Neural Network Representation:

Row 1:



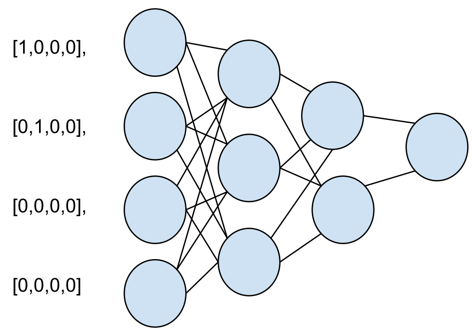
Row 2:



Note: *It will show an error because dimensions are not same,*

To overcome this issue:

- Add zero padding to solve the dimension issue



Disadvantage with using ANN or CNN for sequential data:

- **Input and output dimensions:** In sequential data, the length of input sequences and corresponding output sequences may vary. ANN or CNN requires fixed-size input and output dimensions, leading to dimension mismatch errors.
- **Sparse matrix problem:** One-hot encoding, commonly used for representing categorical variables in ANN or CNN, leads to the creation of large sparse matrices. This results in a significant amount of unnecessary data and computations.
- **Prediction problem:** When predicting future elements in a sequence, it can be challenging to provide the correct input dimension to the ANN or CNN model. The prediction may require a different dimensionality compared to the training dimension, causing difficulties in making accurate predictions.
- **Varying sequence lengths:** Sequential data, such as text, can have varying lengths. ANN or CNN models are not inherently capable of handling inputs with different sizes.
- **Loss of sequential information:** ANN and CNN models process data independently, without considering the order or temporal relationships between elements in the sequence. This can result in the loss of important sequential information.

Note: *That's why in 1980, a network called RNN (Recurrent Neural Network) came into the picture*

To address these challenges, RNN was introduced:

- RNN overcomes the dimension mismatch issue by allowing input sequences of variable lengths and output sequences of different dimensions.
- RNN uses recurrent connections that enable the model to maintain and propagate information across different time steps, preserving sequential information.
- The internal memory of an RNN allows it to process sequential data efficiently and capture dependencies between elements in the sequence.
- RNN models, such as LSTM (Long Short-Term Memory) and GRU (Gated Recurrent Unit), are commonly used for various sequential tasks due to their ability to handle variable-length inputs and retain long-term dependencies.

Sequential Memory and Importance of Previous Data:

Sequential Memory:

- Sequential memory refers to the ability of a model to remember and utilize information from previous elements in a sequence.
- It allows the model to understand and capture dependencies or patterns that exist within the sequential data.
    - **Example: Actual Sequence** - Alphabet
    - Consider the sequence: A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z.
    - Sequential memory helps the model recognize the sequential order and relationship between the letters.
    - **Example: Not Normal Sequence**
    - If we shuffle the sequence randomly: P, B, R, F, X, L, E, A, M, Q, H, Z, W, S, G, N, O, V, C, T, U, I, D, K, J, Y.
    - Sequential memory enables the model to identify that the sequence is not in the expected alphabetical order.
- Sequential data is easy to learn
    - Sequential data, such as the alphabet, often exhibits inherent patterns or dependencies that can be learned and utilized by models with sequential memory.

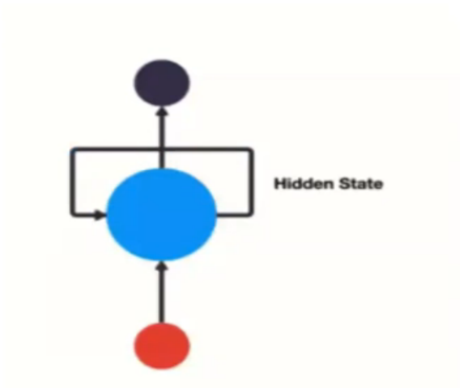
Importance of Previous Data

- In sequential data, the information from previous elements is important for understanding and predicting the current or future elements.
- **For example**, to predict the letter "K" in the sequence "A, B, C, D, E, F, G, H, I, J, K," the model needs to consider the context of the previous letters.
- **Example:**
    - Suppose we want to predict the next letter in the sequence "A, B, C, D, E, F, G."
    - Sequential memory allows the model to understand that the next letter is likely to be "H" based on the sequential pattern of the alphabet.

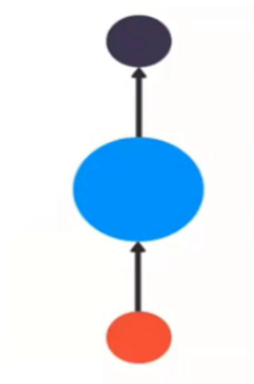
Note: *Sequential data is easy to learn and RNNs are abstract concepts of sequential memory.*

RNN VS ANN:

- RNN incorporates feedback connections to process sequential data and capture temporal dependencies, while ANN does not have feedback connections and is better suited for non-sequential data processing.



RNN



ANN

In RNN, the output is getting applied as input for the next neuron

RNN Output as Input:

- In an RNN (Recurrent Neural Network), the output of a neuron is used as input for the next neuron in the sequence.
- This allows the network to maintain a sense of memory and continuity while processing sequential data.

Unrolling for "Hello":

- To process the word "Hello" in an RNN, it is unrolled or unfolded five times.
- Each unrolled step corresponds to a different time step in the sequence, capturing the progression of the input over time
- **Structure for "Hello" in RNN:**
    - Unrolling Step 1:
        - Input: 'H'
        - Output: 'e'
    - Unrolling Step 2:
        - Input: 'e'
        - Output: 'l'
    - Unrolling Step 3:
        - Input: 'l'
        - Output: 'l'

- Unrolling Step 4:
    - Input: 'l'
    - Output: 'o'

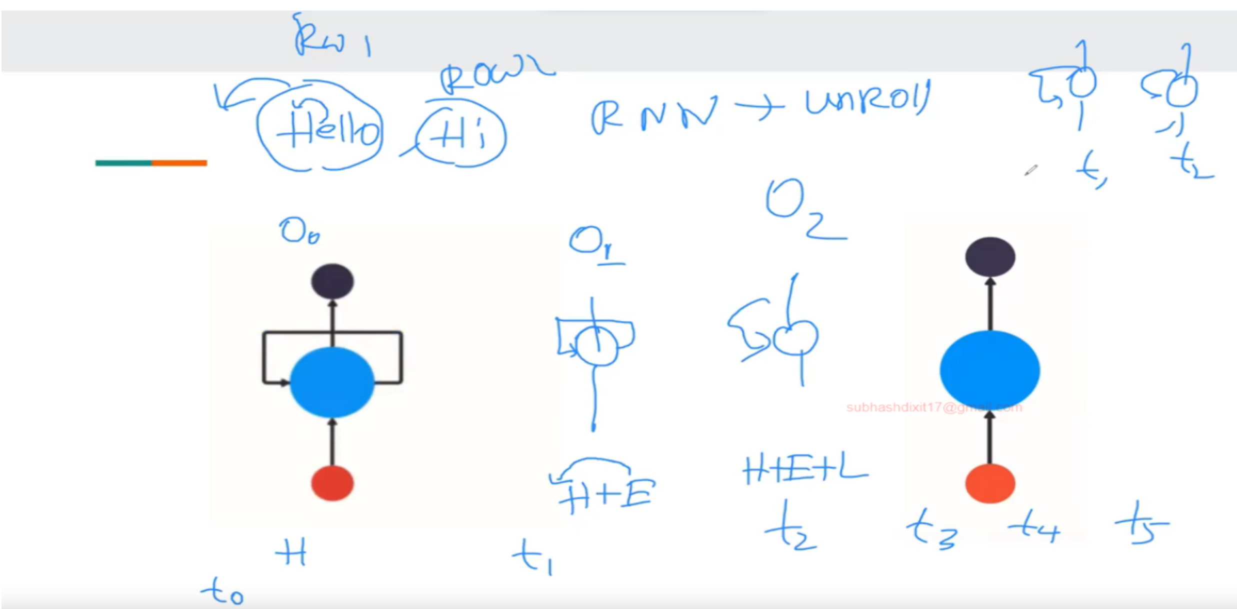
- Unrolling Step 5:
    - Input: 'o'
- Output: (end).

Unrolling for "Hi":

- Similarly, for the word "Hi," it is unrolled or unfolded two times in the RNN.
- Each unrolled step represents a different time step in the sequence, enabling the network to capture the temporal aspect of the input.
- **Structure for "Hi" in RNN:**
    - Unrolling Step 1:
        - Input: 'H'
        - Output: 'i'
    - Unrolling Step 2:
        - Input: 'i'
        - Output: (end)

Time Consideration in RNN:

- RNN takes into consideration the notion of time when creating the network architecture.
- This allows the network to process sequential data and handle variations in input dimensions.



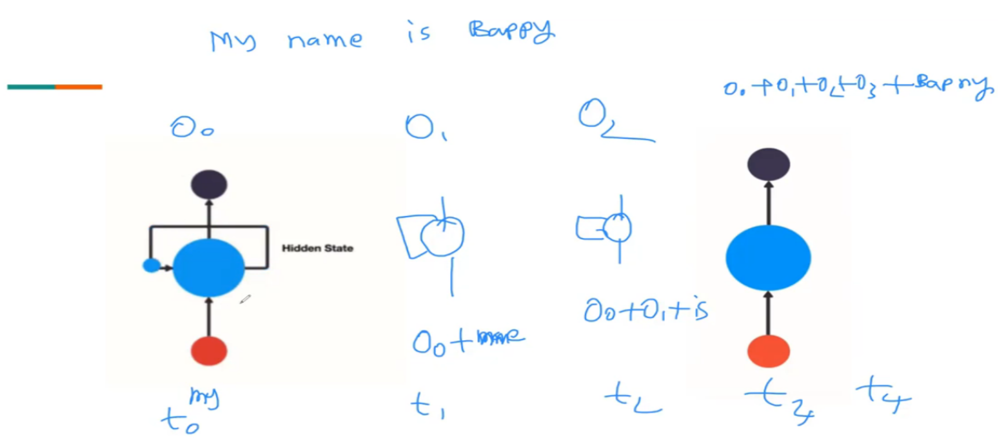
1 Cell RNN Network:

- The RNN network can be represented as a single cell, which processes one input at a time and produces one output.
- The cell has a recurrent connection that allows it to remember and use information from previous time steps.
- However, as the network grows deeper or sequences become longer, RNN faces the problem of vanishing or exploding gradients, causing difficulty in retaining information from distant time steps.

Example:

- 1 Cell Network with Sentence "My name is Bappy":
  - A 1-cell network represents a simplified version of a recurrent neural network (RNN) with a single recurrent cell.
  - It processes one word at a time and maintains an internal state to remember information from previous words.

```
  |--- h1 --|              (output)
  |         |
input       |
  |         |
  |--- h2 --|              (internal state)
```

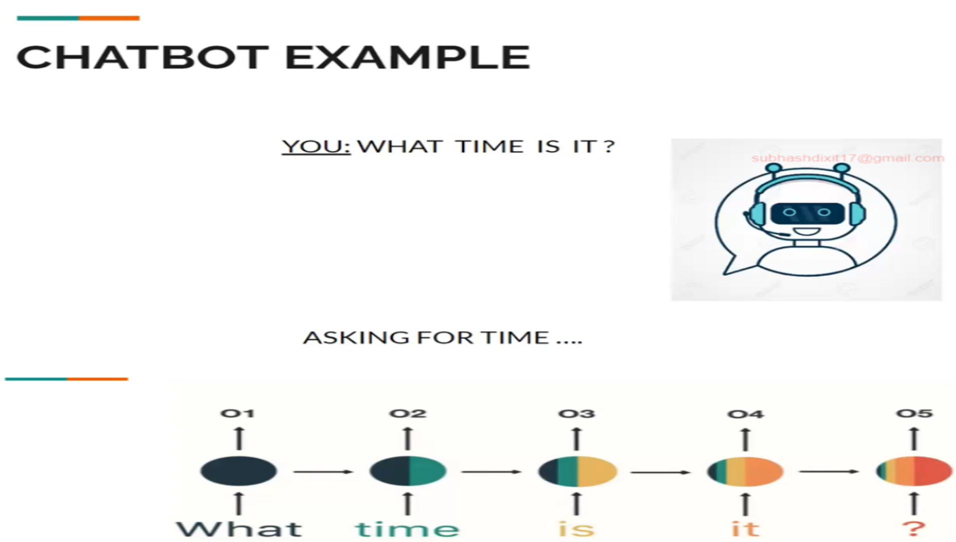


- **Explanation of the Network:**
  - **Input Layer:**
    - Each input represents a single word from the sentence "My name is Bappy."
    - For simplicity, we consider an encoding where each word is represented by a numerical value.
  - **h1 (Output) and h2 (Internal State):**

        - The network has two nodes, h1 and h2, which represent the output and internal state, respectively.
        - h1 receives the input word and produces an output at each time step.
        - h2 represents the internal state, which retains information from previous time steps
    - **Processing Steps:**
        - At the first time step, the input "My" is fed into the network.
        - The network processes the input and updates the values of h1 and h2 accordingly.
        - At the second time step, the input "name" is passed into the network, which utilizes the previous state of h2 to produce the updated output and internal state.
        - This process continues for each subsequent word of the sentence.

Notes on the 1-Cell Network:

- The 1-cell network provides a simplified illustration of how an RNN processes sequential data, specifically with respect to each word in the sentence.
- It shows the flow of information through an input layer, an output node (h1), and an internal state node (h2).
- In reality, RNNs typically have multiple recurrent cells and additional connections for handling longer sequences and capturing more complex dependencies.



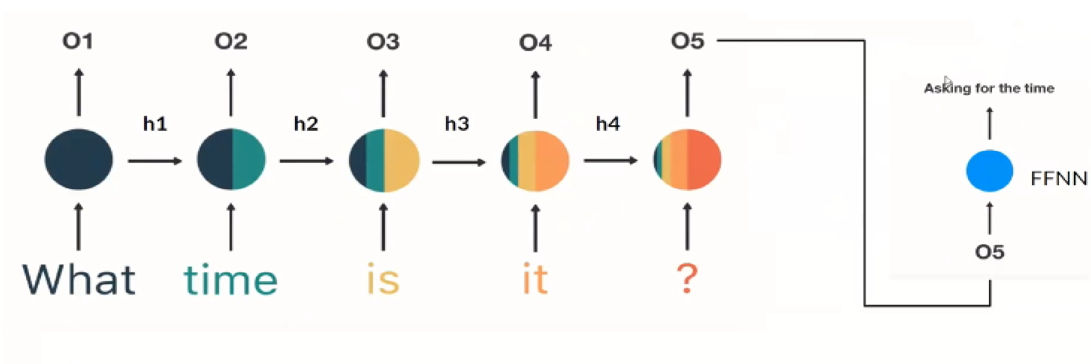
RNN Memory Limitation:

- RNN tends to forget distant past inputs as the network expands, limiting its ability to capture long-term dependencies.

Different Architectures Introduced:

- To overcome the memory limitation, architectures like LSTM, GRU, Bi-Directional LSTM, and Transformer were developed.
- **Advantages of LSTM and GRU:**
    - LSTM and GRU have mechanisms to selectively retain and forget information, enabling them to remember important past inputs effectively.
- **Bi-Directional LSTM:**
    - Bi-Directional LSTM combines information from both past and future time steps, improving the network's understanding of the sequence.
- **Transformer Architecture:**
    - The Transformer architecture addresses the memory limitation by using self-attention mechanisms to capture relationships between different positions in the sequence.
    - **Advantages of Transformer:**
        - Transformers, exemplified by models like GPT3 and GPT4, excel at memorizing and comprehending longer sequences of contextual information.

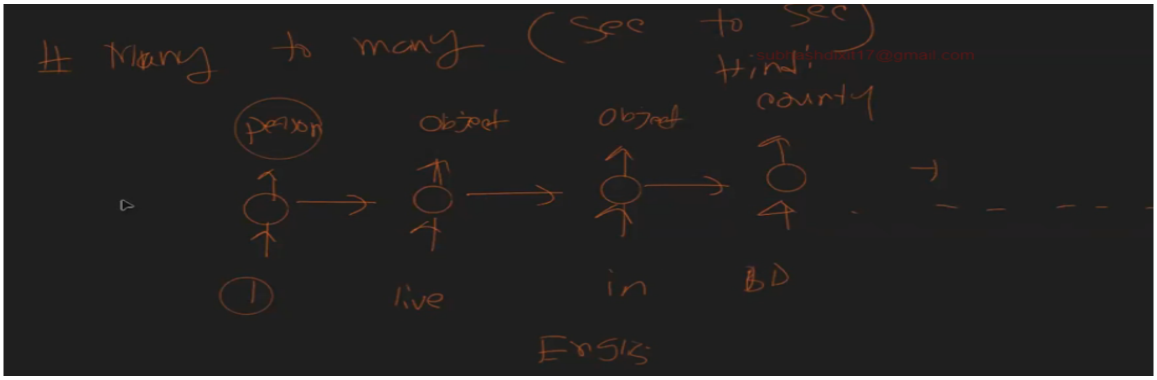
Final Network:



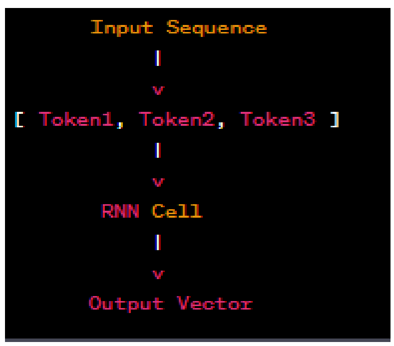
Different types of RNN/Sequential Architecture:

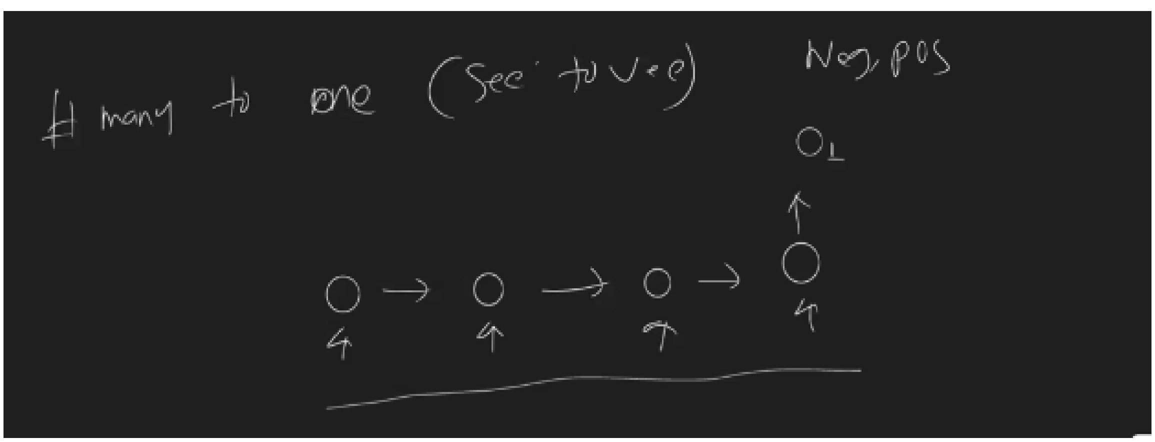
1. **Many to Many (Sequence to Sequence):**
    - Takes a sequence as input and produces a sequence as output.
    - Used for tasks such as language translation or entity recognition, where the input and output are both sequences.
    - **Example:** Language translation, converting English sentences to French sentences.

```
      Input Sequence              Output Sequence
            |                           |
            v                           v
[ Token1, Token2, Token3 ] -> [ Translation1, Translation2, Translation3 ]
            |                           |
            v                           v
       RNN Cell                    RNN Cell
```



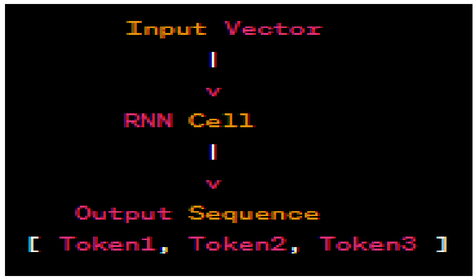
2. **Many to One (Sequence to Vector)**
   - Takes a sequence as input and produces a single vector as output.
   - Used for tasks such as sentiment analysis or spam classification, where the goal is to classify the entire sequence.
   - **Example:** Sentiment analysis of movie reviews, determining whether the review is positive or negative.

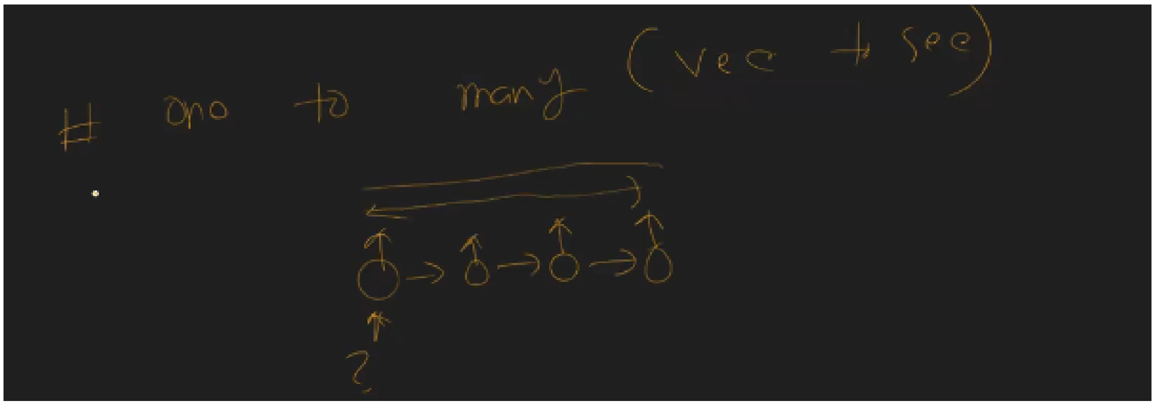




3. **One to Many (Vector to Sequence)**
   - Takes a single vector as input and generates a sequence as output.
   - Used for tasks such as question answering or generating topics based on a given input.

- **Example:** Question answering system that generates a detailed answer based on a given question.





4. **One to One (Vector to Vector):**
   - Takes a single vector as input and produces a single vector as output.
   - Represents a standard feedforward neural network architecture rather than a recurrent structure.
   - Used for tasks where the input and output are both single vectors, without considering sequential dependencies.
   - **Example:** Image classification, where the network takes an image as input and predicts the class label.