# Sequential Data

ANN (Artificial Neural Network) is used for tabular data.

CNN (Convolutional Neural Network) is used for image data.

RNN (Recurrent Neural Network) is a sequential model designed to process sequential data.

**Tabular Data Example**

* Data columns: IQ, marks, gender, placement
* Each entry (IQ, marks, gender) can be represented as independent features.
* In this case, the order of data entries (placement) **does not matter**.

**Sequential Data Example**

* Example: Text data such as "Hi, my name is Nitish."
* Each word is part of a sequence, meaning the order is **important**.
* This is an example of **time series** or sequential data, where the sequence of elements carries information.

**Why Use RNN Instead of ANN for Sequential Data?**

* **Textual Data Varying in Length**:
  + Example inputs:
    - "Hi, my name is Nitish" (5 words)
    - "I love campusX" (3 words)
    - "India won the match" (4 words)
  + **Problem**: Traditional ANNs require **fixed-size input**, which makes handling variable-length text sequences difficult.
* **Why ANN Can't Handle Different Sized Data**:
  + ANNs expect a fixed input size. If inputs have different lengths, they cannot be directly passed to the ANN without adjustments like padding or truncating.
  + This leads to inefficiencies, as variable-length data needs to be transformed to a fixed size.
* **RNN Solution**:
  + RNNs handle **sequential data** effectively by processing one element at a time, maintaining an internal memory to manage variable-length sequences without needing fixed-size inputs.

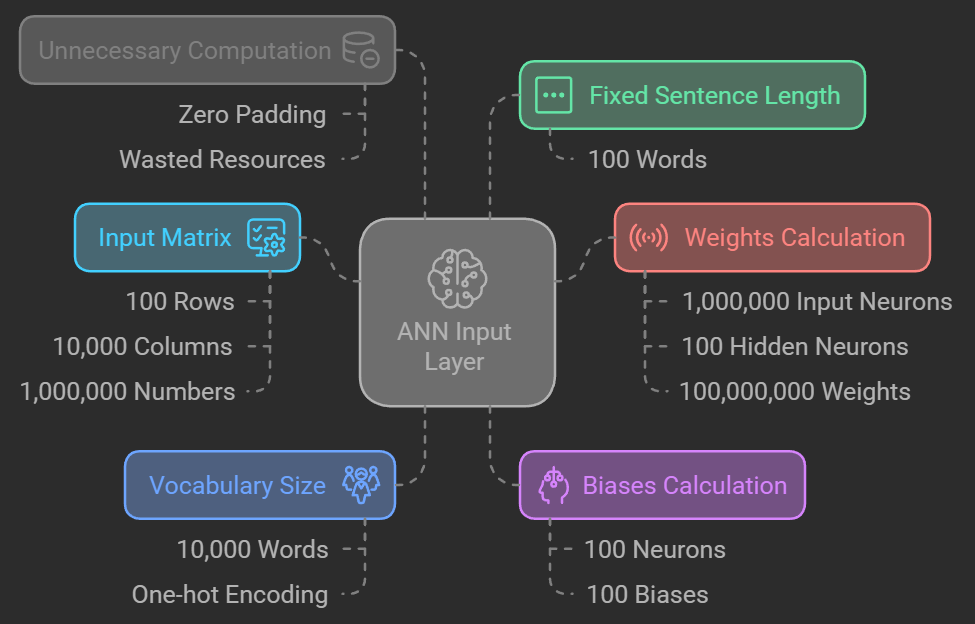
**Why Use RNN Instead of ANN for Sequential Data?**

1. **Input Data Example**:
   * Sentences with different word lengths:
     + "Hi my name is Nitish" (5 words)
     + "I love campusX" (3 words)
     + "India won the match" (4 words)
2. **Vectorization**:
   * Sentences are converted to numerical vectors (e.g., one-hot encoded or word embeddings) for processing in the network.
3. **Challenges with ANN**:
   * **Fixed Input Size**:
     + ANNs require fixed-size inputs, making it hard to handle textual data with varying sentence lengths.
     + Each word in the input must be converted into a fixed-length vector (e.g., 12-dimensional).
     + Example: An ANN with 240 weights would fail when handling input of varying sizes.
   * **Zero Padding**:
     + To ensure all input sentences are of the same length, zero padding is added to shorter sentences, which leads to unnecessary computation.
4. **Vocab Size and Word Limit**:
   * Vocab size can be as large as 10,000 words, but sentence lengths vary.
   * Maximum sentence length can be set to a fixed number (e.g., 100 words), but shorter sentences still need padding, leading to inefficiency.
5. **Inefficient Computation**:
   * Padding results in a lot of unnecessary computations, especially when a sentence contains fewer words than the predefined maximum.
   * For example, if each input is limited to 100 words but most sentences are shorter, a significant amount of zero padding is processed, which is computationally wasteful.
6. **RNN Benefits**:
   * **Sequential Processing**:
     + RNNs can process variable-length sequences directly, word by word, without needing fixed input size or zero padding.
     + They maintain memory of previous inputs, enabling efficient handling of sequential data like text.
7. **Failure of ANN for Sequential Data**:
   * When ANN is forced to handle variable-length data with fixed-size vectors, it leads to failure due to wasted computation and loss of sequential context.

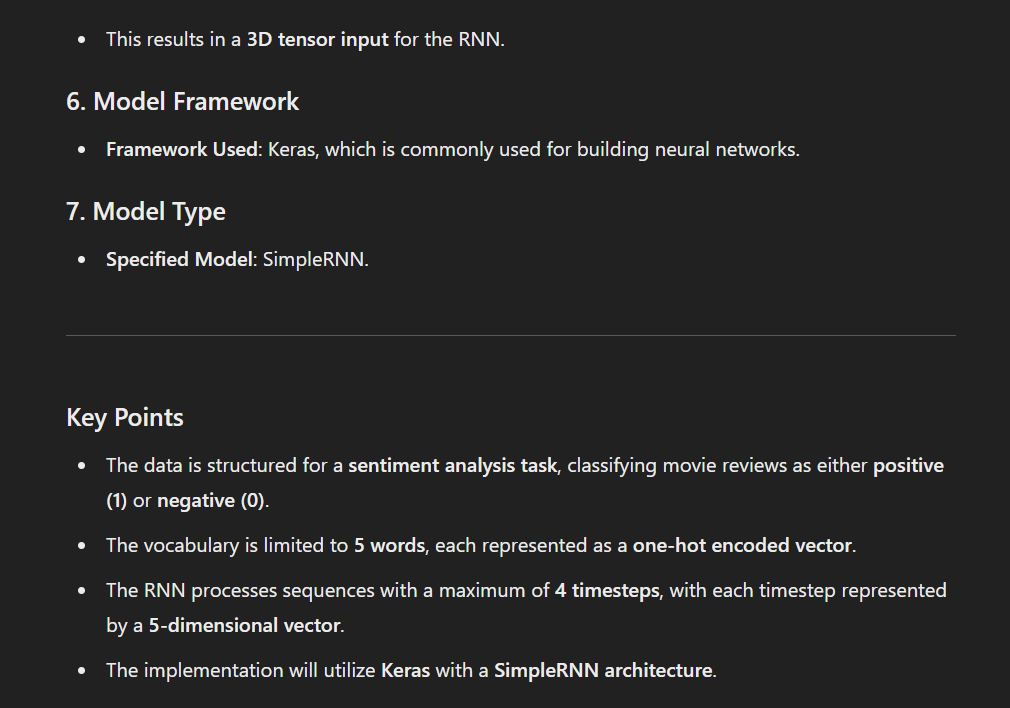
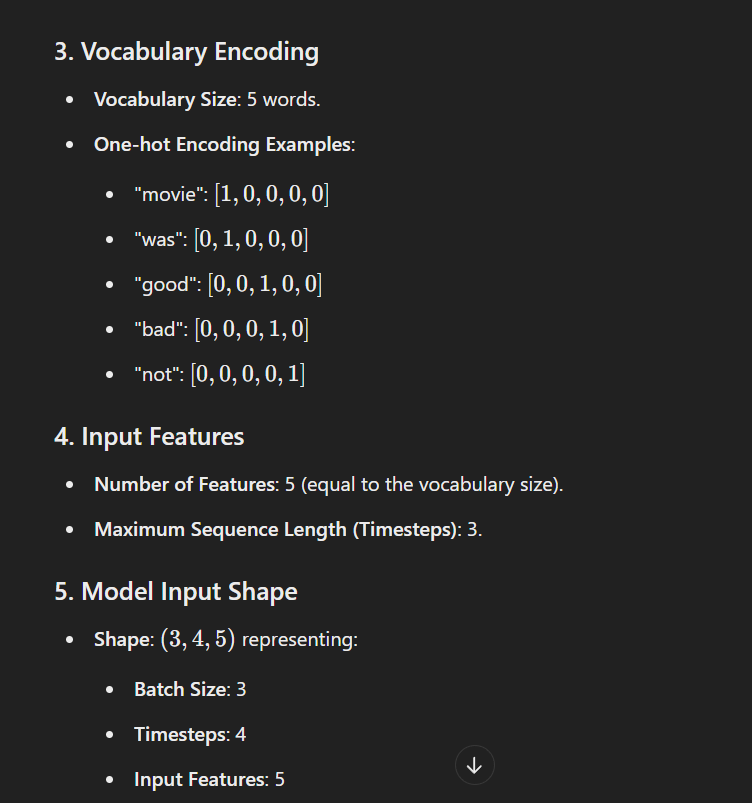
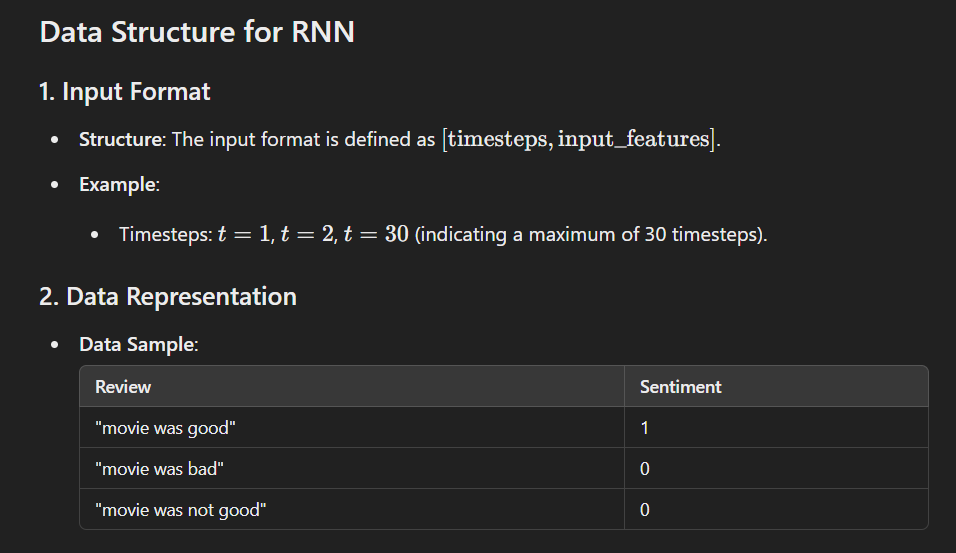
**Summary of Challenges:**

* ANN requires fixed input size → adds zero padding for shorter inputs.
* This padding introduces unnecessary computation and results in inefficient learning.
* RNNs solve this by allowing variable-length input sequences, handling them one at a time.

**Example of 10,000 Words in ANN:**

1. **Vocabulary Size (Vocab Size)**:
   * Consider a vocabulary of **10,000 words**.
   * Each word is represented as a vector of length equal to the vocab size, i.e., **one-hot encoded** vectors of size **10,000** (all values are zero except for one index representing the word).
2. **Fixed Sentence Length**:
   * Suppose we limit the maximum sentence length to **100 words** (even though sentence lengths can vary).
3. **Input Layer**:
   * The input to the ANN would be a matrix where:
     + The number of rows equals the number of words in the sentence (in our case, 100).
     + The number of columns equals the vocab size (in our case, 10,000).
     + Hence, the size of the input matrix = **100 words × 10,000 vocab size = 1,000,000 numbers**.
4. **Weights and Biases in ANN**:
   * For an ANN, each input neuron is connected to every neuron in the next layer. Let's assume the hidden layer has **100 neurons**.
   * **Weights Calculation**:
     + Number of weights between the input layer and the hidden layer = **1,000,000 input neurons × 100 hidden neurons = 100,000,000 weights**.
   * **Biases Calculation**:
     + For each neuron in the hidden layer, there is one bias term. So, for 100 neurons, we have **100 biases**.
5. **Unnecessary Computation**:
   * Due to **zero padding** for shorter sentences (e.g., if a sentence only has 5 words), the ANN will still have to process all **100,000,000 weights** for every sentence, even when most of the input is zero.
   * This makes the model inefficient because much of the computation involves processing zeros (due to padding), leading to unnecessary calculations and wasting computational resources.
6. 

* **Input:** Text data with varying sequence lengths.
* **Padding:** Zero padding is used to make sequences uniform.
* **Prediction:** The goal is to predict based on the text sequences.
* **Sequence Information:** Traditional methods often disregard sequence information.
* **RNN-ANN:** Using an RNN followed by an ANN can leverage sequential information.
* **Output:** The final output is a prediction, such as a numerical value.



# RNN Architecture for Sentiment Analysis

## 1. Input Data Structure

- \*\*Format\*\*: Review | Sentiment

- \*\*Examples\*\*:

- "movie was good" | 1

- "movie was bad" | 0

- "movie was not good" | 0

## 2. Vocabulary Encoding

- \*\*Vocabulary\*\*: 5 words

- \*\*One-hot encoding\*\*:

- movie: [1 0 0 0 0]

- was: [0 1 0 0 0]

- good: [0 0 1 0 0]

- bad: [0 0 0 1 0]

- not: [0 0 0 0 1]

## 3. RNN Structure

- \*\*Input\*\*: 5x3 matrix (5 words, 3 time steps)

- \*\*Hidden Layer\*\*: 3 neurons

- \*\*Output\*\*: 3 neurons (matches hidden layer size)

## 4. Time Steps

- \*\*t=1\*\*: First word input

- \*\*t=2\*\*: Second word input

- \*\*t=3\*\*: Third word input (not shown, but implied)

## 5. Weights and Biases

- \*\*Input to Hidden\*\*: 5x3 weight matrix

- \*\*Hidden to Hidden\*\*: 3x3 weight matrix

- \*\*Hidden to Output\*\*: 3x1 weight matrix

- \*\*Biases\*\*: Added at each layer

## 6. Calculations

- Hidden state: h = tanh(W\_hh \* h\_prev + W\_xh \* x + b\_h)

- Output: y = W\_hy \* h + b\_y

## 7. Total Parameters

- Input weights: 5 \* 3 = 15

- Hidden weights: 3 \* 3 = 9

- Output weights: 3 \* 1 = 3

- Biases: 3 (hidden) + 1 (output) = 4

- Total: 15 + 9 + 3 + 4 = 31 parameters

## Example Walkthrough

1. Input: "movie was good"

- t=1: [1 0 0 0 0] ("movie")

- t=2: [0 1 0 0 0] ("was")

- t=3: [0 0 1 0 0] ("good")

2. At each time step:

- Apply input weights to the current word

- Combine with the previous hidden state

- Apply activation function (tanh)

- Update hidden state

- Pass to the next time step

3. After processing all words:

- Apply output weights to the final hidden state

- Add output bias

- Apply activation function (likely sigmoid for binary classification)

- Produce final sentiment prediction (0 or 1)

The RNN learns to capture the sequential nature of the text, potentially recognizing patterns like "not good" as negative despite "good" being a positive word in isolation.

<https://www.scaler.com/topics/rnn-architecture/>

<https://medium.com/@sachinsoni600517/recurrent-neural-networks-rnn-from-basic-to-advanced-1da22aafa009>