Recurrent Neural Networks

Prof.Somesh Nandi

Department of AIML

Introduction

- In traditional **neural networks**, inputs and outputs are treated independently.
- However, tasks like predicting the next word in a sentence require information from previous words to make accurate predictions.
- To address this limitation, *Recurrent Neural Networks (RNNs)* were developed.

Introduction

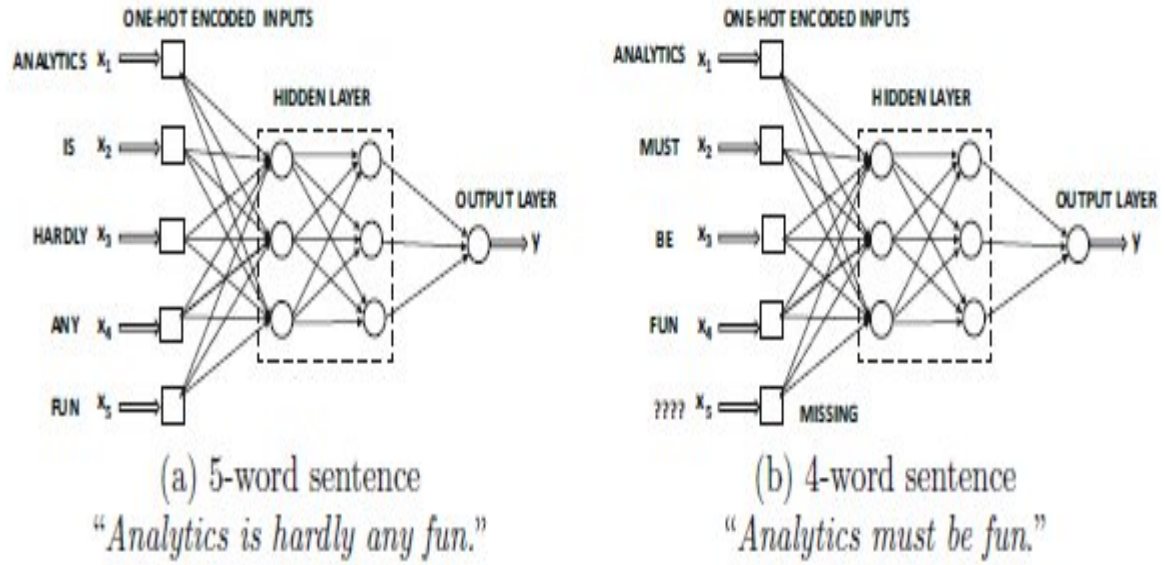
- Certain data types such as time-series, text, and biological data contain sequential dependencies among the attributes.
- In a time-series data set, the values on successive time-stamps are closely related to one another.
- If one uses the values of these time-stamps as independent features, then key information about the relationships among the values of these time-stamps is lost.
- Text is often processed as a bag of words, one can obtain better semantic insights when the ordering of the words is used.
- In such cases, it is important to construct models that take the sequencing information into account.The individual values in a sequence can be either ***real-valued or symbolic***.
- Recurrent neural networks can be used for either type of data.

Challenges of Traditional NN

Bag of Words Approach:

- In some applications, text is processed by treating it as a "bag of words." This means the words in the text are considered without caring about their order. For example, "The cat chased the mouse" and "The mouse chased the cat" would be treated the same because they contain the same words.
- **When Order Matters:**
- This method works fine when the text is long (like a large document) because the overall meaning can often still be understood.
- However, in cases where word order is crucial—like in short text or sentences—this approach fails. The sequence of words changes the meaning. For instance:
- "**The cat chased the mouse**" means the cat is the one doing the chasing.
- "**The mouse chased the cat**" completely flips the meaning, making the mouse the one chasing.

Challenges of Traditional NN



One possible solution is to avoid the bag-of-words approach and create one input for each position in the sequence.

Consider a situation in which one tried to use a conventional neural network in order to perform sentiment analysis on sentences with one input for each position in the sentence.

The sentiment can be a binary label depending on whether it is positive or negative.

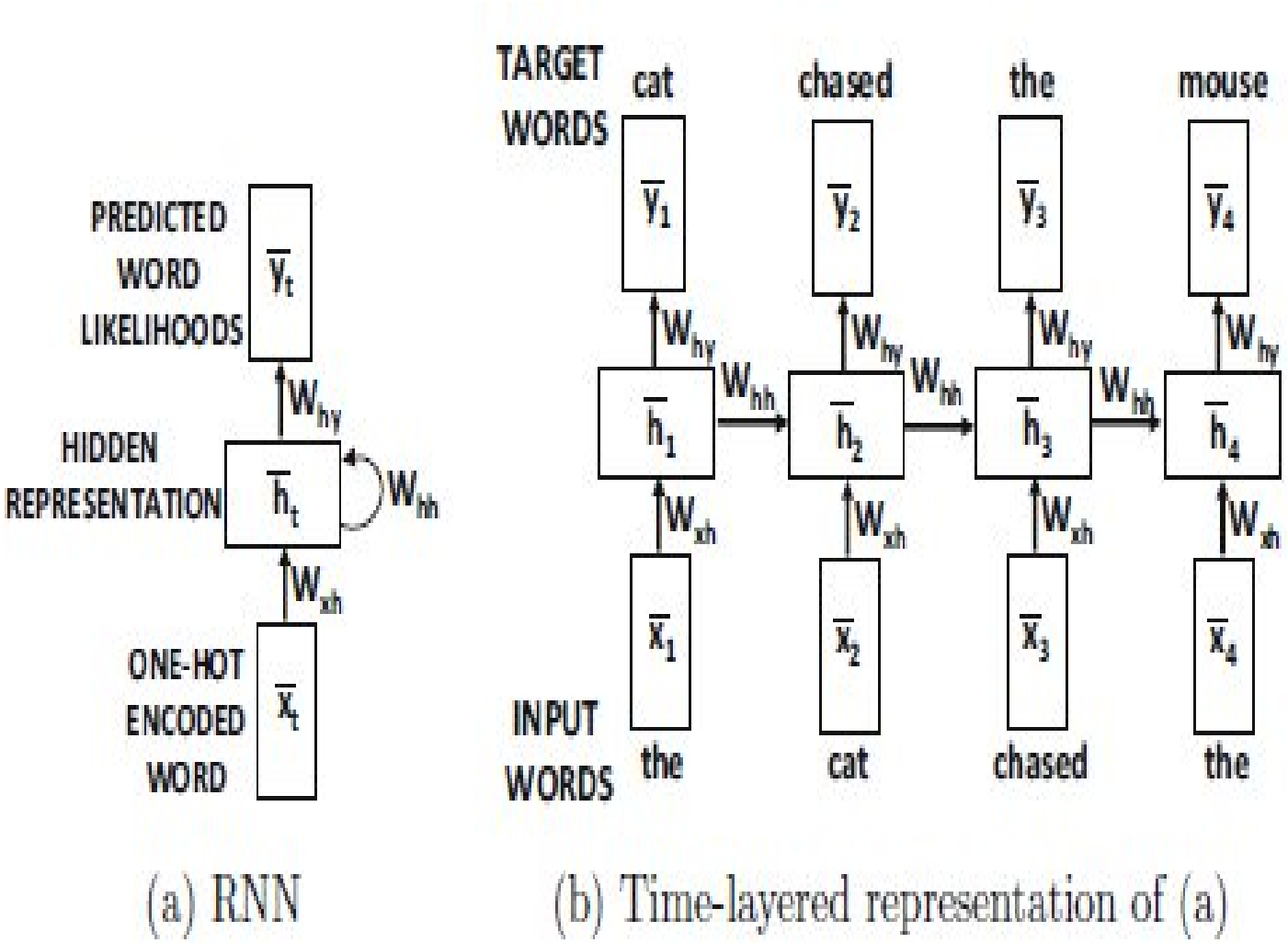
The first problem that one would face is that the length of different sentences is different

Architecture of RNN

- Recurrent Neural Networks introduce a mechanism **where the output from one step is fed back as input to the next, allowing them to retain information from previous inputs.**

- **This design makes RNNs well-suited for tasks where context from earlier steps is essential, such as predicting the next word in a sentence**

Architecture of RNN



(a) RNN (b) Time-layered representation of (a)

Updating the Hidden State in RNNs

1. State Update:

$$h_t = f(h_{t-1}, x_t)$$

where:

- h_t is the current state
- h_{t-1} is the previous state
- x_t is the input at the current time step

2. Activation Function Application:

$$h_t = \tanh(W_{hh} \cdot h_{t-1} + W_{xh} \cdot x_t)$$

Here, W_{hh} is the weight matrix for the recurrent neuron, and W_{xh} is the weight matrix for the input neuron.

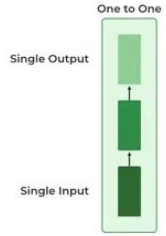
3. Output Calculation:

$$y_t = W_{hy} \cdot h_t$$

where y_t is the output and W_{hy} is the weight at the output layer.

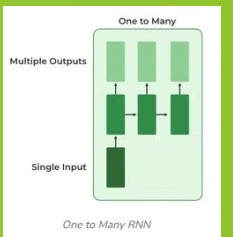
Types of RNN

- **One-to-One RNN :** In this setup, there is a single input and a single output.
- Commonly used for straightforward classification tasks where input data points do not depend on previous elements.



One to One RNN

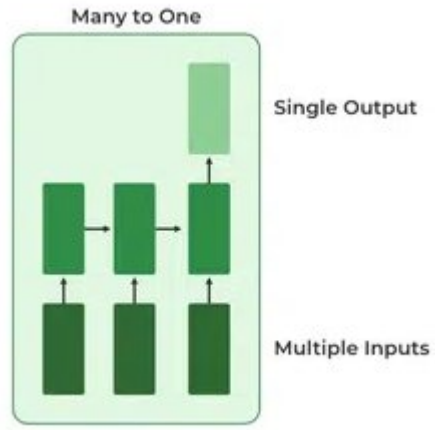
- **One-to-Many RNN**
- In a **One-to-Many RNN**, the network processes a single input to produce multiple outputs over time.
- This setup is beneficial when a single input element should generate a sequence of predictions.
- For example, for image captioning task, a single image as input, the model predicts a sequence of words as a caption.



One to Many RNN

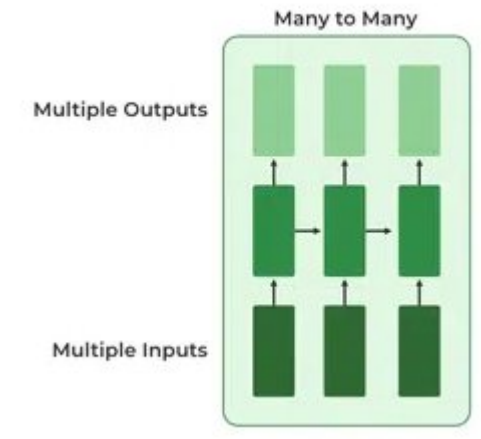
Types of RNN

- **Many-to-One RNN:** The **Many-to-One RNN** receives a sequence of inputs and generates a single output. This type is useful when the overall context of the input sequence is needed to make one prediction.
- **In sentiment analysis, the model receives a sequence of words (like a sentence) and produces a single output, which is the sentiment of the sentence (positive, negative, or neutral).**



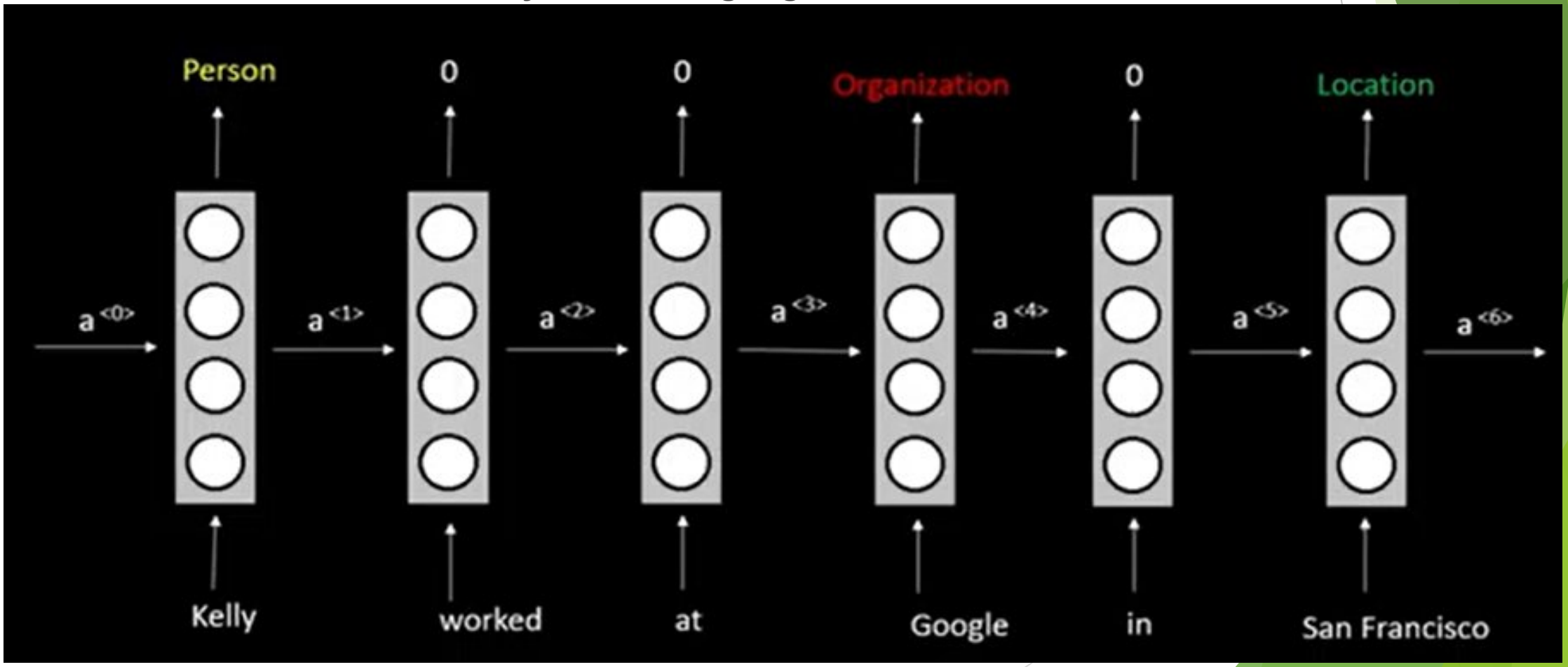
Types of RNN

- **Many-to-Many RNN:** The **Many-to-Many RNN** type processes a sequence of inputs and generates a sequence of outputs.
- This configuration is ideal for tasks where the input and output sequences need to align over time, often in a one-to-one or many-to-many mapping.
- In language translation task, a sequence of words in one language is given as input, and a corresponding sequence in another language is generated as output.



Example -2

- Identify the person, organization and Location (Named Entity Recognition)
  - **Kelly worked at google in San Francisco**



Bi-Directional RNN

A bidirectional RNN processes sequence data in both forward and backward directions.
It uses two RNNs: one that reads the sequence from start to end (forward), and another that reads it from end to start (backward).

- Architecture:
  - Each timestep in the sequence has two hidden states:
    - $\overrightarrow{h_t}$: The forward hidden state.
    - $\overleftarrow{h_t}$: The backward hidden state.
  - The output at each timestep combines these two hidden states (e.g., concatenation or summation).

Bi-Directional RNN

- **Forward Pass:**
  - The forward RNN computes:

$$\overrightarrow{h_t} = \sigma(W_{xh}x_t + W_{hh}\overrightarrow{h_{t-1}})$$

  - The backward RNN computes:

$$\overleftarrow{h_t} = \sigma(W_{xh}x_t + W_{hh}\overleftarrow{h_{t+1}})$$

- **Final Output:**
  - At each timestep t, the combined output is:

$$y_t = f(\overrightarrow{h_t}, \overleftarrow{h_t})$$

    where f is typically concatenation or another aggregation function.

- **Advantages:**
  - **Context Awareness:** Both past and future contexts are considered at each timestep.
  - **Improved Accuracy:** Especially useful in tasks where the entire sequence matters (e.g., speech recognition, language translation).