Natural Language Processing (NLP)

Sequence to Sequence Architecture

Equipping You with Research Depth and Industry Skills

By:

Dr. Zohair Ahmed

Sequence-to-Sequence (Seq2Seq)

- A sequence-to-sequence (Seq2Seq) model is a type of neural network architecture that transforms an input sequence into an output sequence.
- Where both sequences can have different lengths and are often sequential data like text or speech.

Applications of RNN in NLP

- **Autocompletion in Gmail:**
  - Example: When typing "not interested at," Gmail auto-completes with "this time".
  - **RNN** is used to predict and generate the next part of the sentence based on the sequence of words typed so far.
- **Language Translation (e.g., Google Translate):**
  - RNNs are widely used for translating sentences from one language to another by processing each word in a sequence.
- **Named Entity Recognition (NER):**
  - Example: In the sentence "Elon Musk is a billionaire due to Tesla's success," an RNN identifies "Elon Musk" as a person and "Tesla" as a company.
- **Sentiment Analysis:**
  - Example: A product review is analyzed to determine its sentiment (positive/negative), such as "This phone is amazing!" being classified as a positive review.

Why Not Use Simple Neural Networks?

- **Challenges with Standard Neural Networks:**
  - **Sequence Information Loss:**
    a. For instance, "How are you?" vs. "You are how?" – sequence matters in language.
  - **Variable Sentence Lengths:**
    a. Fixed-size input layers cannot handle sentences of varying lengths effectively.
    b. A large fixed-size input layer can lead to inefficiency or unnecessary complexity.
  - **High Computation:**
    a. Converting words to vectors (e.g., using one-hot encoding) increases computational load.

Sequence Modeling and Language Translation

- **Issue with Simple Neural Networks for Translation:**
  - If a sentence's word order is changed, the meaning changes:
    a. "I ate pizza on Sunday" vs. "On Sunday I ate pizza".
  - **ANN Limitations:**
    a. Cannot handle reordering or sequence-sensitive data.
  - **RNN Advantage:**
    a. RNNs remember previous words and context, preserving sequence integrity.

Named Entity Recognition (NER) Example

- **NER Explanation:**
  - In the sentence: "Elon Musk loves SpaceX," an RNN identifies:
    a. "Elon Musk" as a **person**.
    b. "SpaceX" as a **company**.
- **How RNN Works in NER:**
  - Convert words into vectors (e.g., one-hot encoding).
  - Process the sentence word by word, and the RNN maintains the context of previously seen words.

The RNN Architecture

- **Recurrent Structure:**
  - RNNs have a loop structure where the output of each word is fed back into the network for the next word in the sequence.
  - **Time Steps:**
    a. At each step, the output carries forward the context or memory of the sequence processed so far.
- **Example:**
  - For the sentence "Elon Musk loves SpaceX":
    a. The RNN processes "Elon", "Musk", "loves", and "SpaceX" one by one, carrying context at each step.

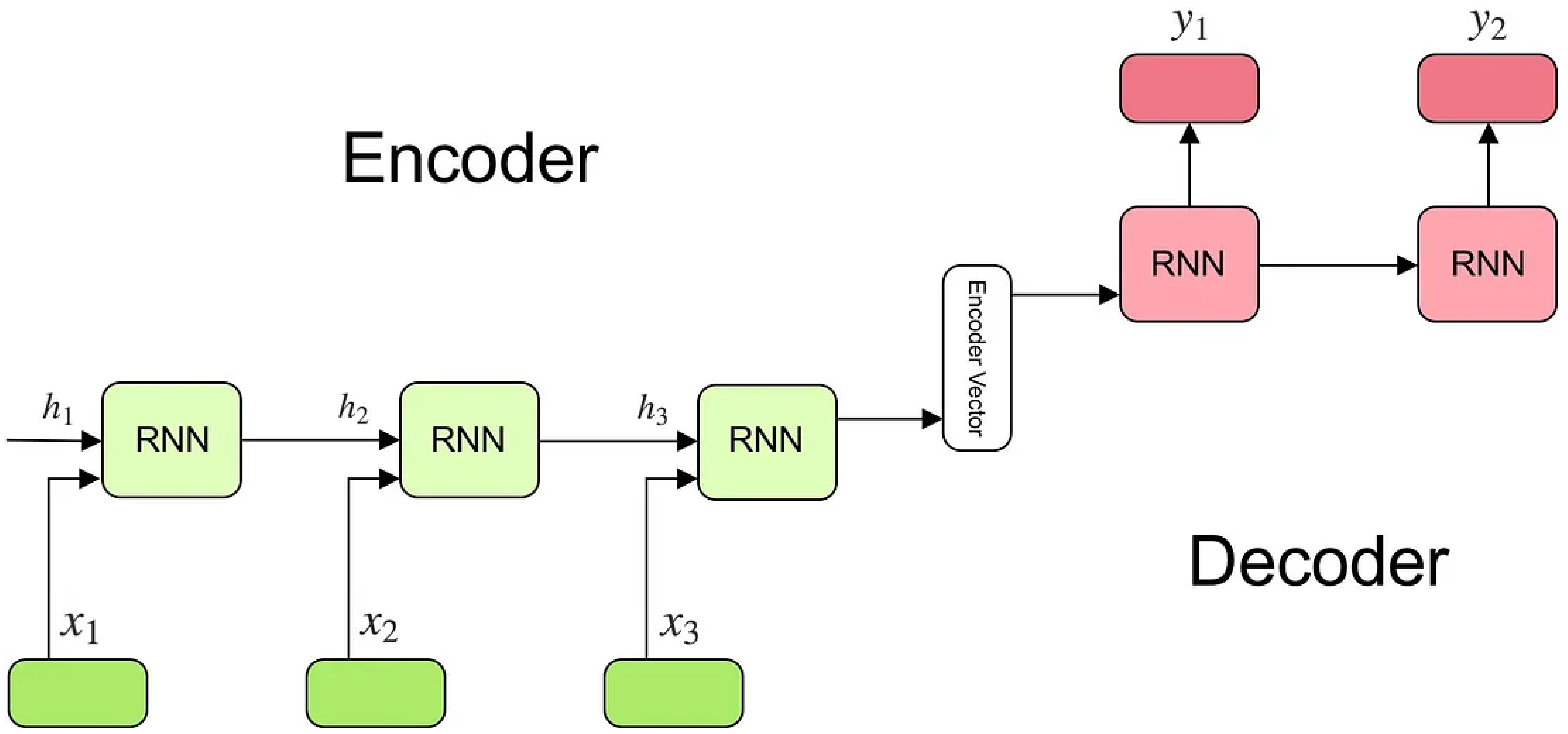
Deep RNN and Training Process

- **Training RNN for NER:**
- **Initial Weights:** Randomly initialized weights.
- **Forward Pass:** Words are processed one by one.
- **Loss Calculation:** Predicted entities are compared to actual labels (e.g., person, company).
- **Backpropagation:** Adjust weights using gradient descent to minimize the loss.
- **Epochs:**
  - Passing the entire dataset through the network multiple times to improve accuracy.

Language Translation with RNN

- **Language Translation with RNN**
- **Encoder-Decoder Architecture:**
  - **Encoder:** Processes the input sequence (source language).
  - **Decoder:** Generates the output sequence (target language).
- **Process:**
  - Input sentence is fed word by word into the encoder.
  - Once the entire sentence is processed, the decoder starts translating the sentence word by word.
- **Deep RNNs:**
  - Can have multiple hidden layers to improve model performance.

How the Sequence-to-Sequence Model works?



Why Use RNN for Sequence Modeling?

- **Key Benefits of RNN:**
  - **Memory and Context:** RNNs have the ability to remember and use past information, crucial for sequence-based tasks like language translation, sentiment analysis, etc.
  - **Handling Variable-Length Sequences:** RNNs can process sequences of varying lengths, making them more flexible than traditional neural networks.

Different Types of Recurrent Neural Networks (RNNs)

- **Introduction to Different Types of Recurrent Neural Networks (RNNs)**
- **Goal**: Understand the different types of RNN architectures used in sequence modeling tasks.
- **Many-to-Many RNN**
- **Many-to-One RNN**
- **One-to-Many RNN**
- **RNN Basics**:
  - RNNs are used for processing sequences where the order of the data matters (e.g., text, music).
  - Key feature: They can handle inputs and outputs of varying lengths.

Many-to-Many RNN – Named Entity Recognition (NER)

- **Use Case**: **Named Entity Recognition (NER)**
  - **Example**: Input sentence: "Elon Musk is the CEO of SpaceX."
    a. **Output**: Tagging of named entities: "Elon Musk" -> Person, "SpaceX" -> Organization.
- **Architecture**:
- Input sequence (e.g., "Elon Musk is the CEO of SpaceX") is processed, and each word is classified (e.g., "Elon" -> Person, "CEO" -> Role, etc.).
- This is a **Many-to-Many** RNN because each word in the input sequence has a corresponding output.
- **Generic Representation**:
  - Inputs: x_1, x_2, ..., x_k (sequence of words).
  - Outputs: y_1, y_2, ..., y_k (tags for each word in the sequence).
- **Other Example**:
  - **Sentence**: "Apple's stock price surged today."
  - **Entity Tags**: "Apple" -> Organization, "stock price" -> Financial Term, "surged" -> Action.

Many-to-Many RNN – Language Translation

- **Use Case**: **Language Translation**
  - **Example**: Input sentence: "I love deep learning."
    a. **Output**: "J'adore l'apprentissage profond." (French translation).
- **Architecture**:
  - The RNN processes the entire input sentence one word at a time, and after processing the last word, the model generates the translation.
  - The number of words in the output can vary from the number of words in the input.
  - **Generic Representation**:
    a. Input sequence (x_1, x_2, ..., x_k) in one language (e.g., English).
    b. Output sequence (y_1, y_2, ..., y_k) in the target language (e.g., French).
- **Translation Example**:
  - Input: "The cat sat on the mat."
  - Output: "Le chat s'est assis sur le tapis."

Many-to-One RNN – Sentiment Analysis

- **Use Case**: **Sentiment Analysis**
  - **Example**: Input paragraph: "This phone is amazing, I love it!"
    a. **Output**: Sentiment rating: Positive (or 5 stars).
- **Architecture**:
  - The RNN processes the entire paragraph (many words) and outputs a single value (the sentiment score).
  - **Generic Representation**:
    a. Input sequence (x_1, x_2, ..., x_k) where each x represents a word.
    b. Output ($\hat{y}$): A single sentiment label (e.g., Positive, Negative, or a numerical score like 1–5 stars).
- **Other Example**:
  - Input: "The movie was boring, it dragged on forever."
  - Output: Negative sentiment (e.g., 1 star).

One-to-Many RNN – Music and Poetry Generation

- **Use Case**: **Music and Poetry Generation**
  - **Example**: Input seed: "Once upon a time" (for poetry generation).
    a. **Output**: "Once upon a time, there lived a king, who ruled a vast kingdom…"
  - **Example for Music**: Input seed note: A simple musical note.
    a. **Output**: A melody created based on the initial seed note.
- **Architecture**:
  - The model is fed a single input (like a seed word or note), and it generates a sequence of outputs.
  - **Generic Representation**:
    a. Input: x_1 (a single word or note).
    b. Outputs: y_1, y_2, ..., y_k (the generated sequence of words or notes).
- **Music Generation Example**:
  - Input: "C"
  - Output: "C, E, G, A, C" (melody formed from the note "C").

Summary of RNN Architectures

- **Many-to-Many RNN**:
  - **Input**: Sequence of words (e.g., sentence or text).
  - **Output**: Sequence of labels (e.g., tags in NER) or translated words.
  - **Examples**: NER, Language Translation.
- **Many-to-One RNN**:
  - **Input**: Sequence of words.
  - **Output**: A single value (e.g., sentiment or classification score).
  - **Examples**: Sentiment Analysis.
- **One-to-Many RNN**:
  - **Input**: A single word or note.
  - **Output**: A sequence of words or notes.
  - **Examples**: Poetry Generation, Music Composition.

Training Process for RNN

- **Training**:
  - **Forward Pass**: The input sequence is passed through the RNN layer by layer.
  - **Loss Calculation**: Predicted output is compared with the actual output, and the loss is calculated.
  - **Backpropagation**: Adjusting the weights to minimize the loss by updating the network's parameters.
- **Optimization**:
  - **Gradient Descent** is used to optimize the model parameters during training.

Training Process for RNN

- **Training**:
  - **Forward Pass**: The input sequence is passed through the RNN layer by layer.
  - **Loss Calculation**: Predicted output is compared with the actual output, and the loss is calculated.
  - **Backpropagation**: Adjusting the weights to minimize the loss by updating the network's parameters.
- **Optimization**:
  - **Gradient Descent** is used to optimize the model parameters during training.

Training Process for RNN

- In an RNN, the **input** at each time step is denoted as $x(t)$, but you are also asking about the **hidden state** $a(t)$, which is an essential component of how RNNs process sequences. Let's break this down clearly:
- **At each time step in an RNN:**
- **Input**: $x(t)$ is the input at time step t.
- **Hidden state**: $a(t)$ is the hidden state at time step t, which carries the memory of the previous time steps.
- **The process of an RNN:**
- At the **first-time step** (t=1), the RNN receives the first input, x(1).
- The RNN starts with an initial hidden state a(0) (this is typically initialized as a vector of zeros or random values).
  - **Hidden State Update**: At time step t=1, the hidden state a(1) is computed using both the input x(1) and the previous hidden state a(0):
- $a(1) = activation\ function(W \cdot [a(0), x(1)] + b)$ Where:
  - W is the weight matrix.
  - *b* is the bias term.
  - The activation function is often a **tanh** or **ReLU** function.
- So, $a(1)$ (the hidden state at time step 1) is computed by taking the initial hidden state $a(0)$ (which is usually zeros) and the input at time step 1, $x(1)$, and combining them through the RNN's weight matrix and activation function.

RNN Unfold