Minor in AI

RNN Cell & Sequence Modelling

March 12, 2025

1 RNN Cell

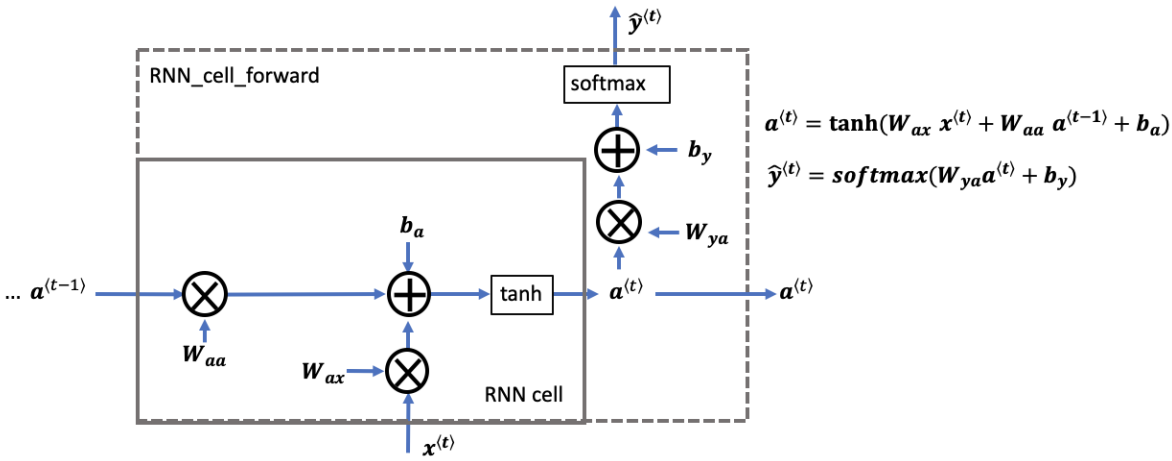


Figure 1: RNN Cell P.C : DeepLearning.Ai

The diagram represents the forward propagation of a basic Recurrent Neural Network (RNN) cell. It takes as input:

- $x^{(t)}$: Input at time step t.
- $a^{(t-1)}$: Hidden state from the previous time step.

1.1 Equations Governing the RNN Cell

The hidden state is updated using the equation:

$$a^{(t)} = \tanh(W_{ax}x^{(t)} + W_{aa}a^{(t-1)} + b_a)$$

where:

- W_{ax} is the weight matrix for input $x^{(t)}$.
- W_{aa} is the weight matrix for the hidden state $a^{(t-1)}$.
- b_a is the bias term.
- $\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$ is the activation function.

The output $\hat{y}^{(t)}$ is computed as:

$$\hat{y}^{(t)} = \text{softmax}(W_{ya}a^{(t)} + b_y)$$

where:

- W_{ya} is the weight matrix for the output.
- b_y is the bias term for the output.
- Softmax ensures that the output is a probability distribution.

1.2 Flow of Data

1. The previous hidden state $a^{(t-1)}$ is multiplied by W_{aa}.
2. The current input $x^{(t)}$ is multiplied by W_{ax}.
3. The sum of these, along with bias b_a, is passed through the tanh activation to obtain $a^{(t)}$.
4. The hidden state $a^{(t)}$ is used to compute the output $\hat{y}^{(t)}$.
5. Softmax activation is applied to obtain the final output distribution.

Note

NER model that we have discussed is a *Many - to - Many* model.
The model being discussed is a **Many-to-Many** model, meaning that both the input and output sequences have the same size. This type of model is often used in sequence prediction tasks such as machine translation or speech-to-text systems.

Movie Ratings

- **Input:** Review text.
- **Output:** A numerical rating in the set $\{1, 2, 3, 4, 5\}$.

In the context of movie ratings, the mapping is **Many-to-One**, meaning that a sequence of words (review) is mapped to a single output (rating). This is typical for sentiment analysis and rating prediction models.

2 Sequence Modeling

Sequence modeling plays a crucial role in many AI applications, particularly in Natural Language Processing (NLP) and time-series analysis. The architectures can be classified into five primary types, as shown in the image below.

Each rectangle represents a vector, and arrows denote transformations (such as matrix multiplications). The input vectors are in red, the output vectors in blue, and the recurrent states in green. Below, we explore different sequence modeling paradigms.

2.1 One-to-One (Feedforward Network)

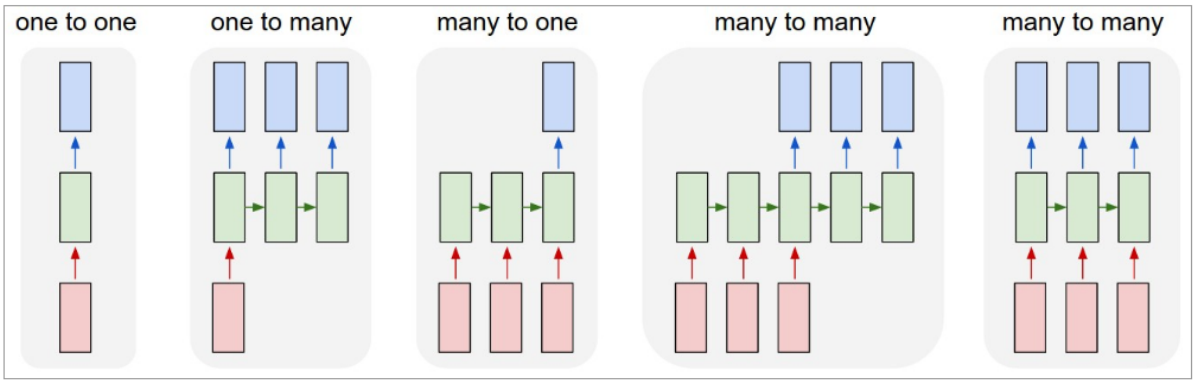
Description: This is the standard mode of processing, where a fixed-size input is mapped to a fixed-size output using a feedforward network.

Example: Image classification, where an image is passed through a deep neural network to produce a label.

2.2 One-to-Many

Description: A single input generates a sequence of outputs. This is common when generating sequential data from a fixed representation.

Example: Image captioning, where a single image input is processed to generate a sequence of words describing the image.



Each rectangle is a vector and arrows represent functions (e.g. matrix multiply). Input vectors are in red, output vectors are in blue and green vectors hold the RNN's state (more on this soon). From left to right: **(1)** Vanilla mode of processing without RNN, from fixed-sized input to fixed-sized output (e.g. image classification). **(2)** Sequence output (e.g. image captioning takes an image and outputs a sentence of words). **(3)** Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment). **(4)** Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French). **(5)** Synced sequence input and output (e.g. video classification where we wish to label each frame of the video). Notice that in every case are no pre-specified constraints on the lengths sequences because the recurrent transformation (green) is fixed and can be applied as many times as we like.

Figure 2: Types of Sequence Modeling P.C: Karpathy

2.3 Many-to-One

Description: A sequence of inputs is mapped to a single output. This is useful in scenarios where a final decision or classification is based on sequential information.

Example: Sentiment analysis, where a sentence is processed to determine whether its sentiment is positive, negative, or neutral.

2.4 Many-to-Many (Different Input and Output Lengths)

Description: Both input and output are sequences, but they do not necessarily have the same length. The model must learn to transform an input sequence into a corresponding output sequence.

Example: Machine translation, where an input sentence in one language is translated into an output sentence in another language.

2.5 Many-to-Many (Synced Sequences)

Description: Input and output are both sequences of the same length. This is useful when each input element corresponds directly to an output element.

Example: Video classification, where each frame in a video is labeled with a category.

3 Encoder – Decoder Architecture

The **Encoder-Decoder model** is widely used in **sequence-to-sequence (Seq2Seq)** tasks, such as:

- Machine Translation (e.g., English to French translation)
- Text Summarization

- Speech Recognition
- Chatbots and Conversational AI

It consists of two main components:

- **Encoder:** Processes the input sequence and converts it into a fixed-length vector (also called the *context vector*).
- **Decoder:** Takes the encoded vector and generates the output sequence step by step.

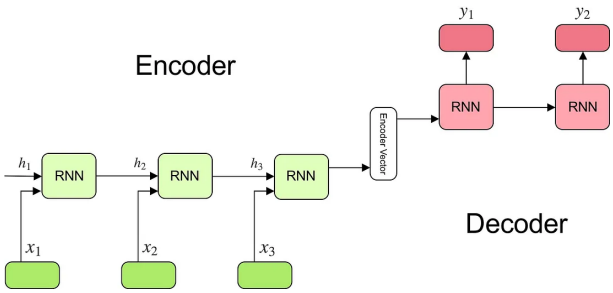


Figure 3: Encoder - Decoder Architecture

3.1 Working Mechanism

3.1.1 Encoder

- The encoder consists of a stack of **Recurrent Neural Networks (RNNs)**, such as LSTMs or GRUs.
- It processes the input sequence $X = (x_1, x_2, ..., x_T)$, where each x_t represents a token (word, character, or feature) at timestep t.
- The encoder updates its hidden state at each step:

$$h_t = f(Wx_t + Uh_{t-1}) \tag{1}$$

  where W and U are weight matrices, and f is an activation function (e.g., tanh).
- After processing the entire sequence, the final hidden state h_T is passed as the **context vector** to the decoder.

3.1.2 Decoder

- The decoder is another RNN that takes the encoded context vector and generates an output sequence $Y = (y_1, y_2, ..., y_N)$. At each step, the decoder predicts the next token using the previous hidden state and output:

$$y_t = g(Vh_t + Wy_{t-1}) \tag{2}$$

  where V and W are weight matrices, and g is a nonlinear function (e.g., softmax for classification).
- The decoder generates tokens one at a time until a special *end-of-sequence* token is reached.

3.1.3 Example: Machine Translation (English to French)

- Suppose we have an English sentence: *"I love cats."*
- The encoder processes each word and converts it into a fixed-length representation.
- The decoder then generates the translated French sentence: *"J'aime les chats."* step by step.

3.2 Word Translation Problem

The encoder-decoder model is widely used for various NLP tasks such as machine translation, speech-to-text, and text summarization. It consists of two main parts:

1. **Encoder:** Processes the input sequence and encodes it into a fixed-length vector.
2. **Decoder:** Uses the encoded vector to generate the output sequence step-by-step.

Consider the sentence:

I like eating Pongal

Each word in the sentence is represented as x_1, x_2, x_3, x_4. The encoder processes these inputs and generates a context vector (encoder vector), which is then passed to the decoder. The decoder then generates the translated words sequentially.

Since we are dealing with a word translation problem, the length of input and output sequences is the same. However, the decoder must choose each output word from a large vocabulary. To accomplish this, we apply a **softmax** function at each decoding step to select the most probable word.

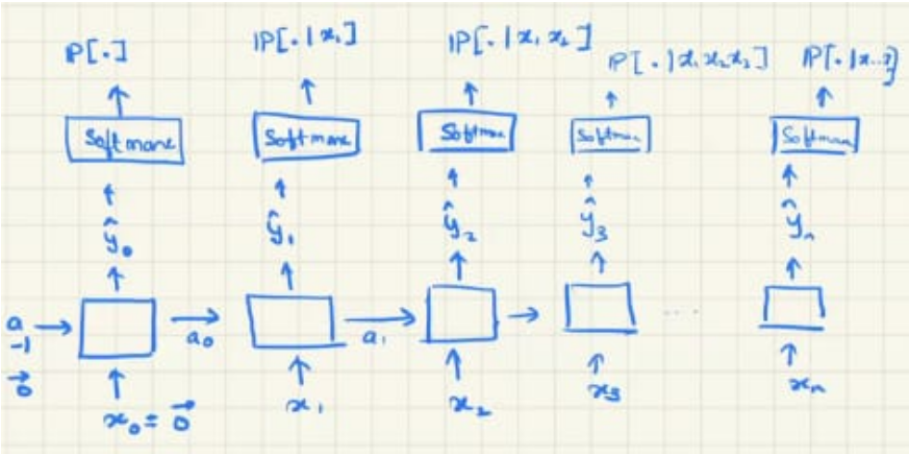


Figure 4: Architectural Workflow

Consider a vocabulary of 10 words. The decoder's output at each step should be one of these words. The softmax function helps assign a probability distribution over the vocabulary and selects the most likely word.

4 Speech-to-Text Processing

Another common application of sequence-to-sequence models is speech-to-text conversion.

4.1 Example: Sentence Occurrence Probability

Consider two similar sentences:

1. The apple & pair salad was good.
2. The apple & pear salad was good.

If sentence 2 occurs more frequently in a given corpus than sentence 1, then:

$$O(S_2) > O(S_1) \tag{3}$$

where $O(S_i)$ represents the occurrence of sentence S_i in the corpus. Given a total of N sentences in the corpus, the probability of each sentence occurring is given by:

$$\frac{O(S_2)}{N} > \frac{O(S_1)}{N} \tag{4}$$

Thus, language models learn from such occurrences and help in selecting the most probable words during speech recognition and machine translation.

5 Key Takeaways

1. RNN cells process sequential data by maintaining hidden states, but suffer from vanishing gradient issues, making them ineffective for long-term dependencies.
2. Different types of sequence modeling tasks, such as one-to-one, one-to-many, many-to-one, and many-to-many, address various real-world problems like machine translation and speech recognition.
3. Encoder-Decoder architectures enable handling variable-length input and output sequences by compressing input into a context vector before generating output.
4. Speech-to-text models convert audio signals into text using feature extraction techniques like MFCC, followed by sequence modeling using RNNs or Transformers.
5. Advanced architectures like attention mechanisms improve the performance of sequence models by focusing on relevant parts of the input, overcoming limitations of simple RNNs.