Definition

Recurrent neural networks (**RNNs**) are a class of artificial neural networks designed for processing sequential data, such as text, speech, and time series, where the order of elements is important. Unlike feedforward neural networks, which process inputs independently, RNNs utilize recurrent connections, where the output of a neuron at one time step is fed back as input to the network at the next time step. This enables RNNs to capture temporal dependencies and patterns within sequences.

Type to enter text

"RNNs can be designed with different input and output lengths, making them suitable for a variety of applications.

★ illustrates the different forms RNNs can take depending on the input and output lengths

one to many

Input: A single input, like an image vector (from a CNN) or a seed word/ vector.

Output: A sequence of words, such as a sentence describing the image ("A dog is playing in the park") or a series of notes in a musical composition.

NLP Example:

- *Text Generation:* Generating text from a keyword or topic. You input a single vector representing a general subject (e.g., "sports") and the RNN generates a paragraph about sports.

Many-to-one

Input: A sequence of words (e.g., a sentence or a document).

Output: A single output, such as a sentiment score (positive, negative, neutral) or a category label (sports, politics, technology).

NLP Examples:

- *Sentiment Analysis:* Analyzing a customer review ("This product is amazing!") and outputting a sentiment score indicating positive sentiment.
- *TextClassification:* Classifying an email as spam or not spam based on its content.

Many-to-many

Input: A sequence of words.

Output: A sequence of labels, one for each input word.

NLP Examples:

- *Part-of-Speech (POS) Tagging:* Assigning a grammatical tag to each word in a sentence (e.g., "The/DET dog/NOUN is/ VERB running/VERB").
- *Named Entity Recognition (NER):* Identifying named entities in a text and classifying them (e.g., "John/PERSON Smith/ PERSON works/VERB at/PREP Google/ ORGANIZATION").
-

Core Concepts:

- **Hidden State (h):** This acts as the 'memory' of the RNN, storing information about past inputs. The hidden state at the current time step t, denoted as h_t, is calculated based on the previous hidden state h_{t-1} and the current input x_t.
- **Weight:** These values determine how much influence the input x_t and the previous hidden state h_{t-1} have on the current hidden state h_t. W_x is the weight matrix for the input layer, and W_h is the weight matrix for the previous hidden state.
- **Activation Function:** This non-linearly transforms the calculated values, allowing the model to learn more complex patterns. In this case, the tanh function is used.
- **Batch Size:** The number of data samples processed at once. Here, the batch size is 1, meaning only one data sample is processed at a time.
- **Dimension:** Represents the size of a vector or matrix. d is the dimension of the word vector, and D_h is the dimension of the hidden state.

Explanation of the Equations:

- **Hidden Layer:** $h_t = \tanh(W_x x_t + W_h h_{t-1} + b)$
- **$W_x x_t$:** Multiplies the input x_t by the weight matrix W_x. This represents how the input information is reflected in the current hidden state.
- **$W_h h_{t-1}$:** Multiplies the previous hidden state h_{t-1} by the weight matrix W_h. This represents how the past information influences the current hidden state.
- **b:** This is the bias term. It helps the model avoid being biased in a particular direction.
- **tanh(...):** Passes the values calculated above through the tanh activation function, transforming them into values between -1 and 1.
-
- **Output Layer:** $y_t = f(W_y h_t + b)$
- **$W_y h_t$:** Multiplies the current hidden state h_t by the weight matrix W_y. This represents how the hidden state information is reflected in the output.
- **b:** This is the bias term. It helps the model avoid being biased in a particular direction.
- **f(...):** Passes the values calculated above through a non-linear activation function f to obtain the final output.

Input x_t (4x1): The word vector at the current time step t. Each word is represented by 4 numbers.

2 **Weight W_x (4x4):** A matrix that transforms the input x_t to match the size of the hidden state. It transforms 4 input values into 4 hidden state values.

3 **Weight W_h (4x4):** A matrix that reflects the previous hidden state h_{t-1} in the current hidden state. It transforms 4 previous hidden state values into 4 current hidden state values.

4 **Previous Hidden State h_{t-1} (4x1):** The hidden state at the previous time step t-1. It contains past information.

5 **Bias b (4x1):** A vector that adjusts the model to prevent it from being biased in a particular direction.

6 **Wxxt (4x1):** The result of the matrix multiplication of W_x and x_t. It represents how much the input information is reflected in the hidden state.

7 **Whht-1 (4x1):** The result of the matrix multiplication of W_h and h_{t-1}. It represents how much the past information is reflected in the hidden state.

8 **Wxxt + Whht-1 + b (4x1):** The sum of the values calculated above.

9 **tanh(Wxxt + Whht-1 + b) (4x1):** The value obtained by passing the value calculated above through the tanh activation function, resulting in the current hidden state h_t.

Using Keras

Initiation

```
from tensorflow.keras.layers import SimpleRNN

model.add(SimpleRNN(hidden_units))
```

★

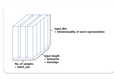
```
# When using additional arguments
model.add(SimpleRNN(hidden_units, input_shape=(timesteps, input_dim)))

# Alternative notation
model.add(SimpleRNN(hidden_units, input_length=M, input_dim=N))
```

★

- **hidden_units**: Defines the size of the hidden state. It's also the same as the size of the value (output dimension, output_dim) that the memory cell sends to the next memory cell in the sequence and to the output layer. Think of it as increasing the capacity of the RNN. For small to medium-sized models, common values are typically 128, 256, 512, or 1024.
- **timesteps**: This is also referred to as the length of the input sequence (input_length). It represents the number of time steps in the sequence.
- **input_dim**: The size (or dimensionality) of the input at each time step.

RNN Layer

RNN layers receive a 3D tensor of size (batch_size, timesteps, input_dim) as input. batch_size refers to the number of data samples processed at once during training



How does the RNN layer output the hidden state after receiving the input 3D tensor?

The RNN layer emits two types of outputs depending on the user's settings. If you want to return only the hidden state of the memory cell at the final time step, it returns a 2D tensor of size (batch_size, output_dim). However, if you want to collect the hidden state values of each time step of the memory cell and return the entire sequence, it returns a 3D tensor of size (batch_size, timesteps, output_dim). This can be set by setting True to the return_sequences parameter of the RNN layer.

Final Hidden State (2D Tensor): Shape

(batch_size, output_dim)

Sequence of Hidden States (3D Tensor): Shape:

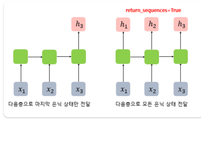
(batch_size, timesteps, output_dim)

return_sequences Parameter: The return_sequences parameter in the RNN layer controls which output mode is used.

- return_sequences=False (default): Returns only the final hidden state (2D tensor).
- return_sequences=True: Returns the sequence of hidden states (3D tensor).

output_dim is hidden_units: The output_dim (the dimensionality of the hidden state vectors) is determined by the hidden_units parameter that you set when defining the RNN layer. They are the same thing.

- **return_sequences=False**: The RNN reads the entire sequence until the end, it gives you a single summary vector representing the meaning of the whole sentence. This is useful for tasks like sentiment analysis.
- **return_sequences=True**: The RNN reads the sentence word by word, and at each word, it gives you a vector representing the meaning *up to that point* in the sentence. This is useful for tasks like machine translation or named entity recognition, where you need to process the sequence step by step.



shows the difference between setting return_sequences = True and not setting it (or setting it to False) when time step=3.

If return_sequences=True is selected, the memory cell outputs the hidden state value for every time step.

If not specified separately, or if selected as return_sequences=False, the memory cell outputs only one hidden state value. And this single value is the hidden state value of the memory cell at the last time step.

Code

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import SimpleRNN

model = Sequential()
model.add(SimpleRNN(3, input_shape=(2,10)))
# model.add(SimpleRNN(3, input_length=2, input_dim=10))와 동일함.
model.summary()
```

Output

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn_1 (SimpleRNN)     (None, 3)                 42
=================================================================
Total params: 42
Trainable params: 42
Non-trainable params: 0
_________________________________________________________________
```