

Introduction to Time Series Forecasting

**Zeham Management Technologies
BootCamp by SDAIA
August 19th, 2024**



SDAIA
الهيئة السعودية للبيانات
والذكاء الاصطناعي
Saudi Data & AI Authority

Introduction to Time series

Let's start together...





Agenda

Time series analysis

Time series components

Time series models

RNN's Architecture

Preprocessing for Time Network Analysis

Evaluation Metrics

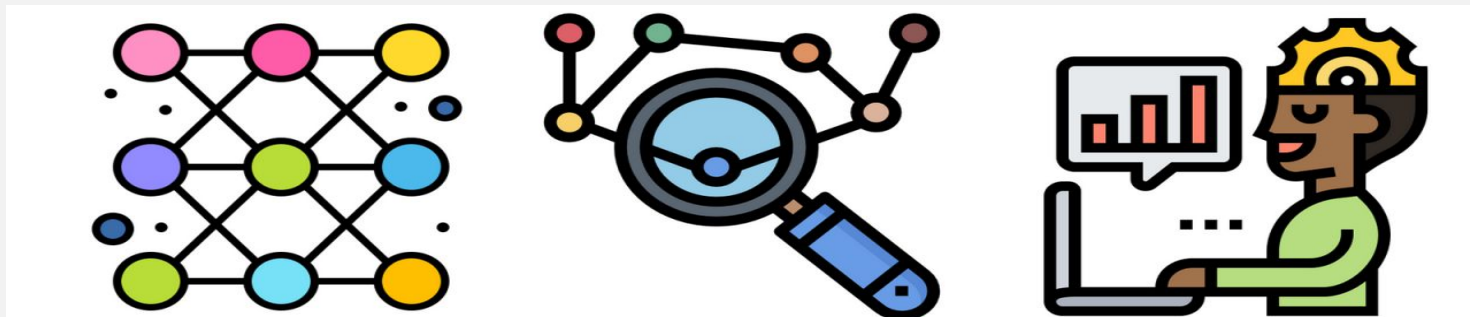


Recurrent Neural Network



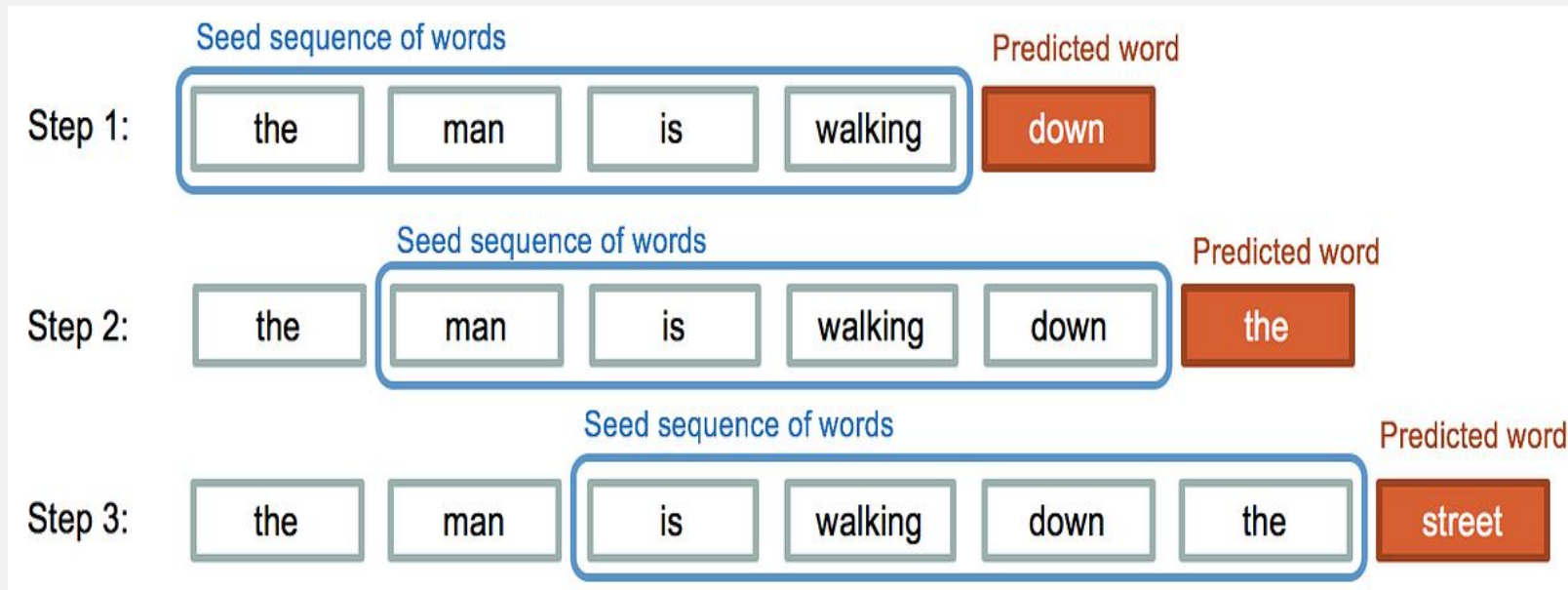
Introduction to Recurrent Neural Networks

- A Deep Learning approach for modelling sequential data is Recurrent Neural Networks (RNN).
- RNNs generalize to other than sequential data, such as geographical or graphical data, because of its design.
- RNNs are formed from feedforward networks but can process sequential data in a way that other algorithms can't.





Introduction to Recurrent Neural Networks (cont'd)

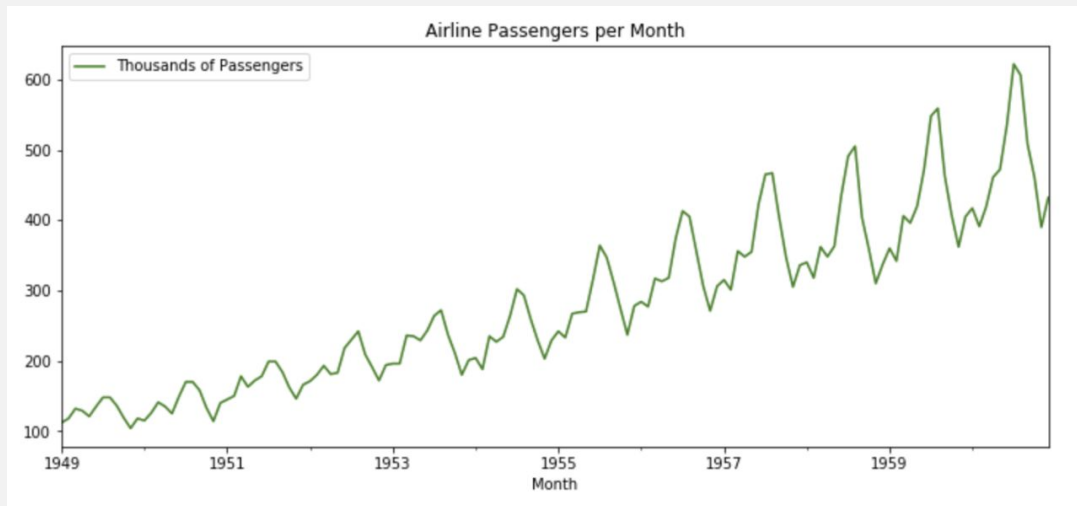


Time series analysis

Time Series

1. Definition. & Significance

A Time Series is defined as a series of data points indexed in time order. The time order can be daily, monthly, or even yearly. Given below is an example of a Time Series that illustrates the number of passengers of an airline per month from the year 1949 to 1960.

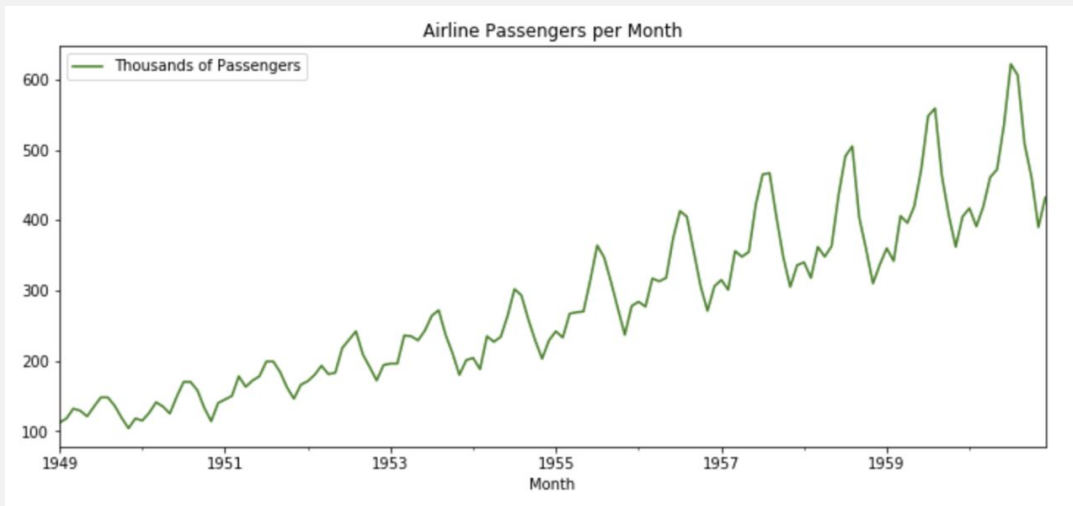


Time Series

1. Definition. & Significance

Time Series Forecasting

Time Series forecasting is the process of using a statistical model to predict future values of a time series based on past results.



Time Series analysis

1. Definition. & Significance

Time series analysis is a specific way of analysing a sequence of data points collected over an interval of time. In time series analysis, analysts record data points at consistent intervals over a set period of time rather than just recording the data points intermittently or randomly.

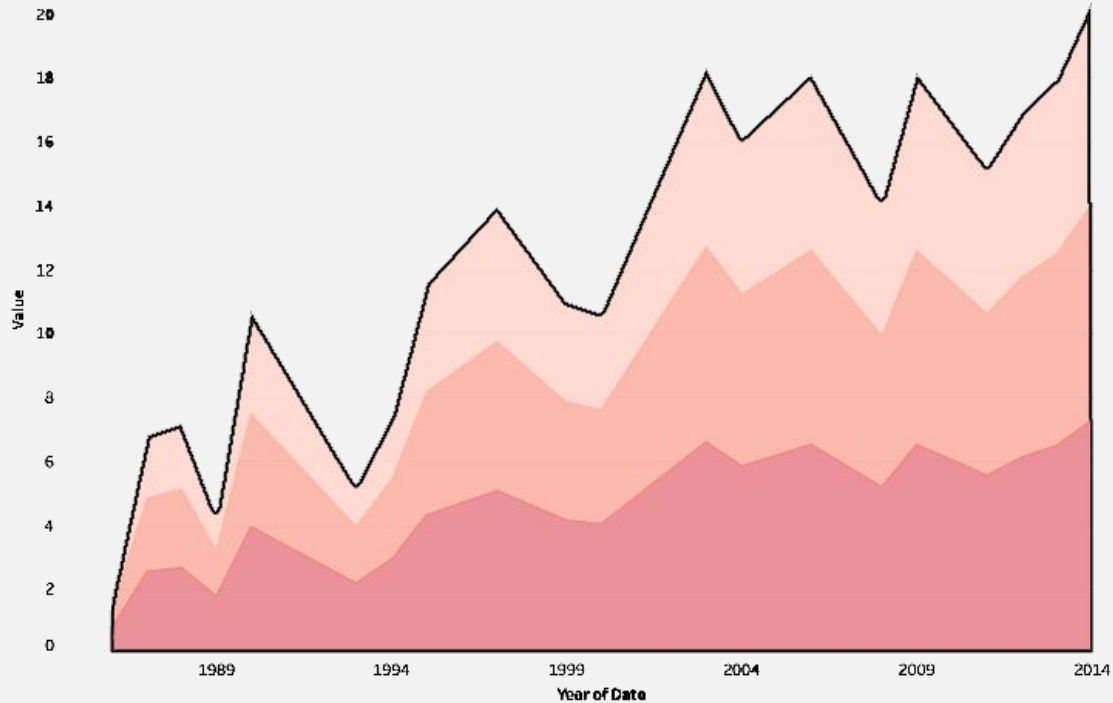
Time series analysis is critical for businesses to predict future outcomes, assess past performances, or identify underlying patterns and trends by studying patterns over time, organizations can understand past performance and predict future in a relevant and actionable way. . Time series helps turn raw data into insights companies can use to improve performance and track historical outcomes.



Time Series analysis

Temperature Example :

Average Temperature

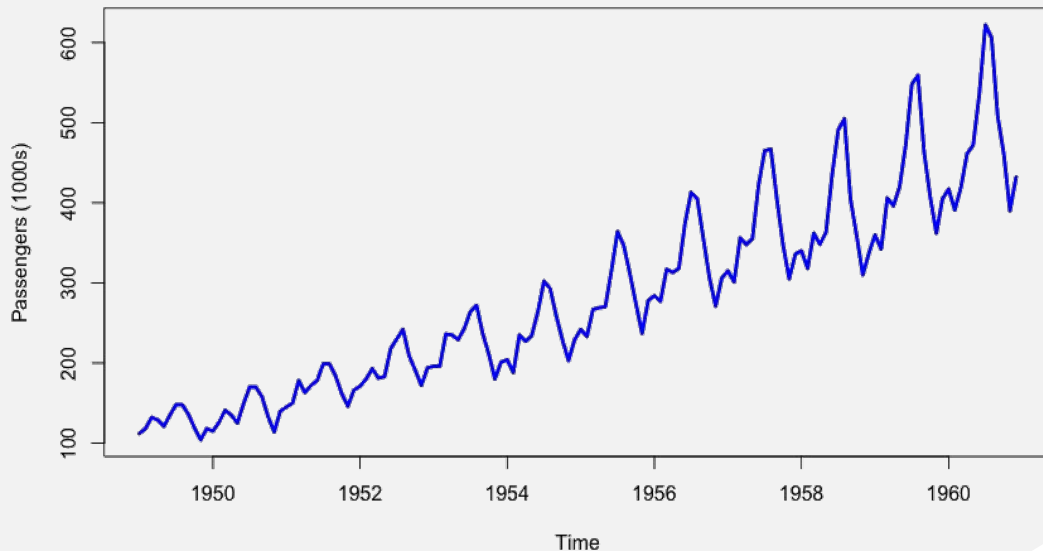


Time Series analysis

2. Applications.

Forecasting has a range of applications in various industries. It has tons of practical applications including:

1. Weather forecasting.
2. Healthcare forecasting.
3. Finance forecasting.
4. Business forecasting.
5. Social studies forecasting.



Time Series analysis

3. Some use cases.

1. To predict the number of incoming or churning customers.
2. To explaining seasonal patterns in sales.
3. To detect unusual events and estimate the magnitude of their effect.
4. To Estimate the effect of a newly launched product on number of sold units.



Time series components

Time series components

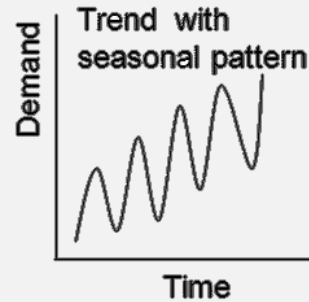
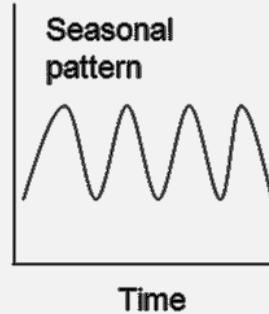
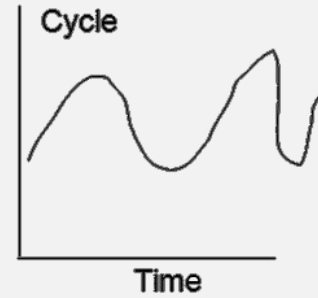
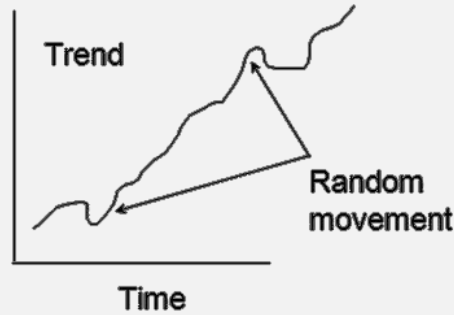
The various reasons or the forces which affect the values of an observation in a time series are the components of a time series.

The four categories of the components of time series are:

- **Trend**
- **Seasonal Variations**
- **Cyclic Variations**
- **Irregular Variation**
- **ETS Decomposition**



Time series components



[SOURCE](#)



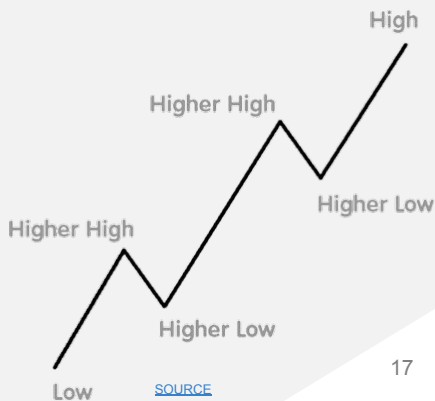


Time series components

Trend

Trends show the general direction of the data, and whether it is increasing, decreasing, or remaining stationary over an extended period of time. Trends indicate the long-term movement in the data and can reveal overall growth or decline. Trend can be :

- **upward .**
- **downward .**
- **linear .**
- **non-linear.**

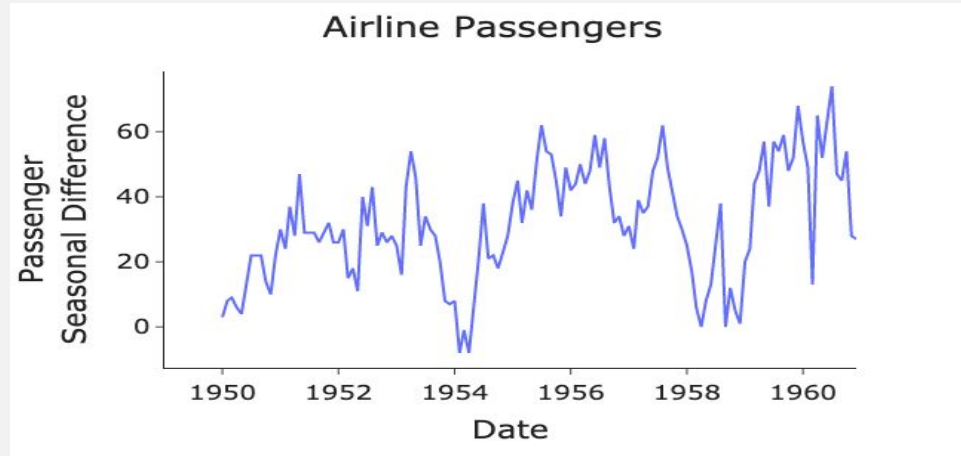




Time series components

Seasonal Variations

Seasonality refers to predictable patterns that recur regularly, like yearly retail spikes during the holiday season. Seasonal components exhibit fluctuations fixed in timing, direction, and magnitude. For instance, electricity usage may surge every summer as people turn on their air conditioners.



[SOURCE](#)

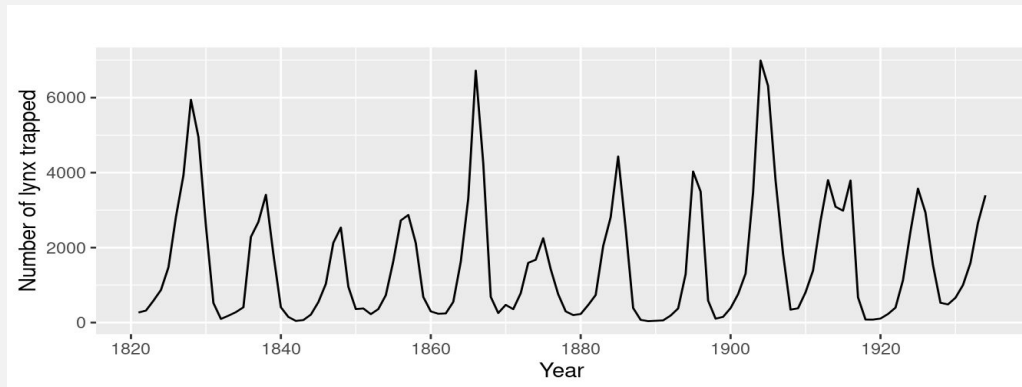




Time series components

Cyclic Variations

Cycles demonstrate fluctuations that do not have a fixed period, such as economic expansions and recessions. These longer-term patterns last longer than a year and do not have consistent amplitudes or durations. Business cycles that oscillate between growth and decline are an example.



[SOURCE](#)

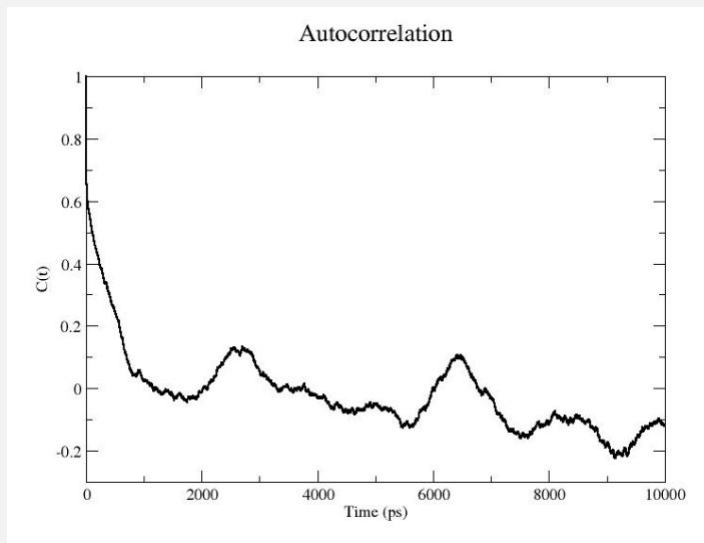




Time series components

Irregular Variation

Irregular These are the fluctuations in the time series data which become evident when trend and cyclical variations are removed. These variations are unpredictable, erratic, and may or may not be random.

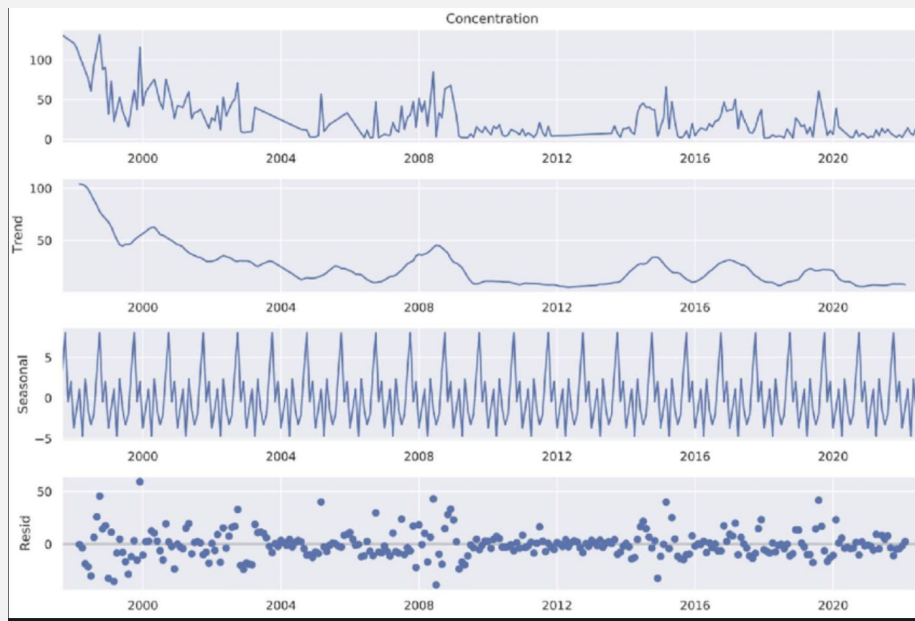




Time series components

ETS Decomposition

ETS Decomposition is used to separate different components of a time series. The term ETS stands for Error, Trend, and Seasonality.



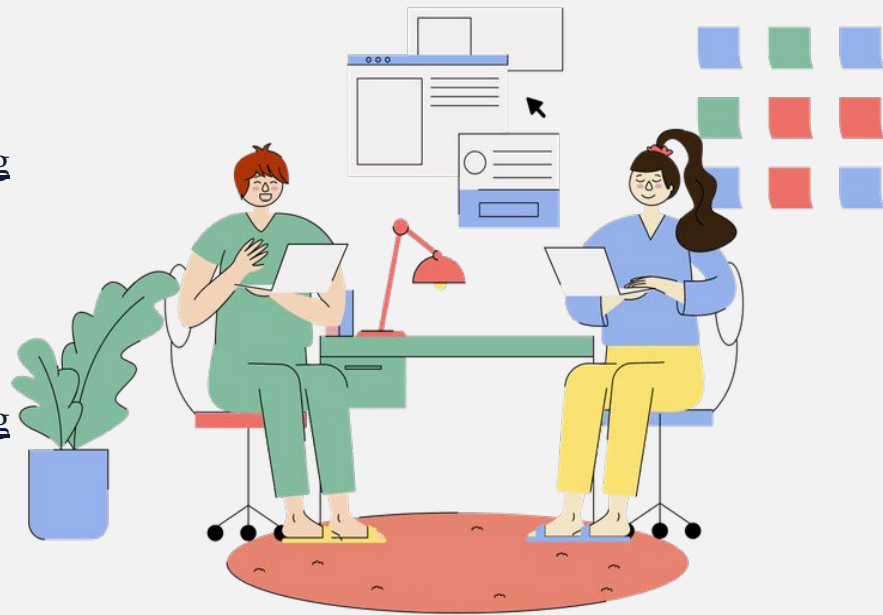
▶ Let's Practice

Tutorial:

- [5-Introduction to time series/2-Time Series Forecasting \(RNN\)/LAB/ETS_Tutorial.ipynb](#)

Dataset:

[5-Introduction to time series/2-Time Series Forecasting \(RNN\)/LAB/Datasets//content/airline-passengers.csv](#)



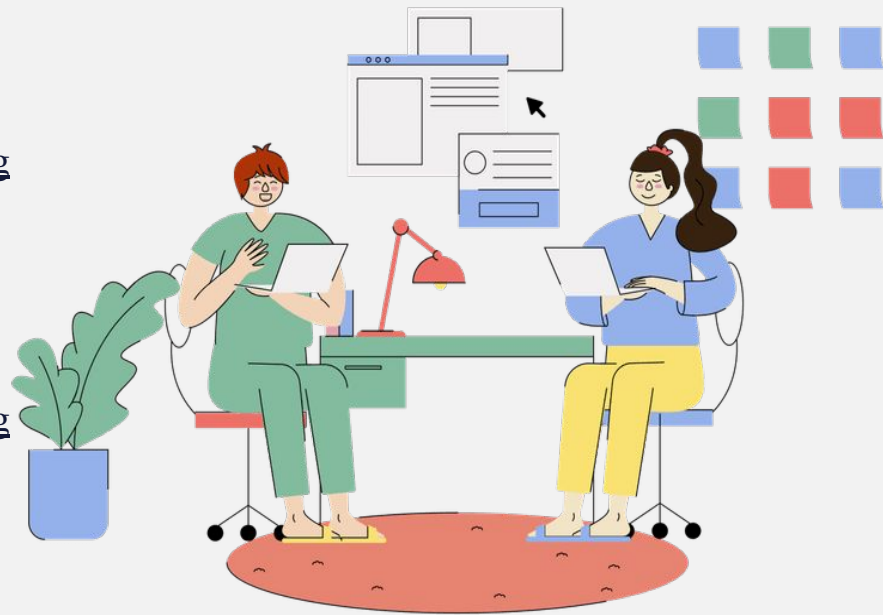
▶ Let's Practice

Exercise:

- [5-Introduction to time series/2-Time Series Forecasting \(RNN\)/LAB/ETS_Exercise.ipynb](#)

Dataset:

- [5-Introduction to time series/2-Time Series Forecasting \(RNN\)/LAB/Datasets/traffic.csv](#)



Time series models

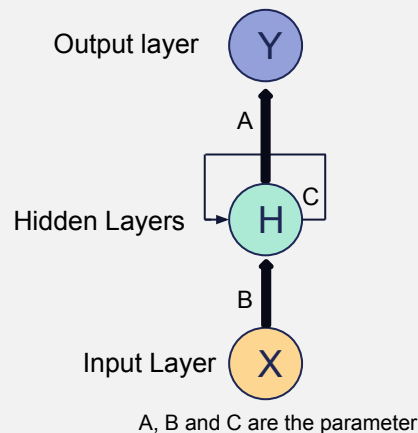
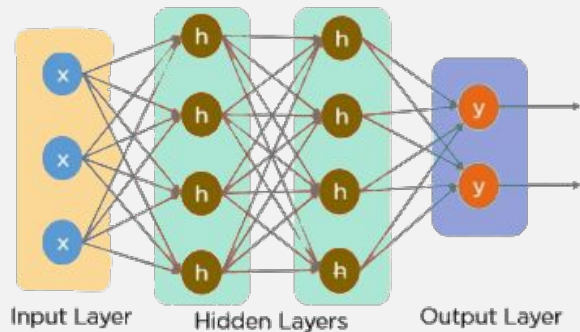
Time series models

Recurrent Neural Network

What is a Recurrent Neural Network (RNN)?

RNN works on the principle of saving the output of a particular layer and feeding this back to the input in order to predict the output of the layer.

Below is how you can convert a Feed-Forward Neural Network into a Recurrent Neural Network:



Time series models

Recurrent Neural Network

Here's a breakdown of its key components:

- **Input Layer:** This layer receives the initial element of the sequence data. For example, in a sentence, it might receive the first word as a vector representation.
- **Hidden Layer:** The heart of the RNN, the hidden layer contains a set of interconnected neurons. Each neuron processes the current input along with the information from the previous hidden layer's state. This "state" captures the network's memory of past inputs, allowing it to understand the current element in context.
- **Activation Function:** This function introduces non-linearity into the network, enabling it to learn complex patterns. It transforms the combined input from the current input layer and the previous hidden layer state before passing it on.
- **Output Layer:** The output layer generates the network's prediction based on the processed information. In a language model, it might predict the next word in the sequence.



Time series models

Types of Recurrent Neural Network

One to One :

- This type of neural network is known as the Vanilla Neural Network .

One to Many :

- This type of neural network has a single input and multiple outputs.

Many to One :

- This RNN takes a sequence of inputs and generates a single output.

Many to Many :

- This RNN takes a sequence of inputs and generates a sequence of outputs.





Time series models

Recurrent Neural Network

Two Issues of Standard RNNs :

- Vanishing Gradient Problem

Vanishing gradient problem is a phenomenon that occurs during the training of deep neural networks, where the gradients that are used to update the network become extremely small or "vanish" as they are backpropogated from the output layers to the earlier layers.

- Exploding Gradient Problem

While training a neural network, the gradients keep on getting larger and larger as the backpropagation algorithm progresses. This, in turn, causes very large weight updates and causes the gradient descent to diverge.



► Time series models

Applications of Recurrent Neural Network

Recurrent neural networks (RNNs) shine in tasks involving sequential data, where order and context are crucial.

Let's explore some real-world use cases. Using RNN models and sequence datasets, you may tackle a variety of problems, including :

- **Music Generation**
- **Video Captioning**
- **Sentiment Analysis**
- **Stock Market Recommendation**
- **Speech Recognition**
- **Machine Translation**
- **Text Generation**
- **Time Series Forecasting**



Time series models

Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) architecture designed to address the vanishing gradient problem and capture long-term dependencies in sequential data.

We use this formula to understand LSTM :

$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$



Time series models

Long Short-Term Memory (LSTM)

The LSTM architecture is based on the following key components:

1. **Cell State (C^t):** This represents the memory of the LSTM and can store information over long sequences. It can be updated, cleared, or read from at each time step.
2. **Hidden State (H^t):** The hidden state serves as an intermediary between the cell state and the external world. It can selectively remember or forget information from the cell state and produce the output.
3. **Input Gate (i^t):** The input gate controls the flow of information into the cell state. It can learn to accept or reject incoming data.
4. **Forget Gate (f^t):** The forget gate determines what information from the previous cell state should be retained and what should be discarded. It allows the LSTM to “forget” irrelevant information.
5. **Output Gate (o^t):** The output gate controls the information that is used to produce the output at each time step. It decides what part of the cell state should be revealed to the external world.



► Time series models

Long Short-Term Memory (LSTM)

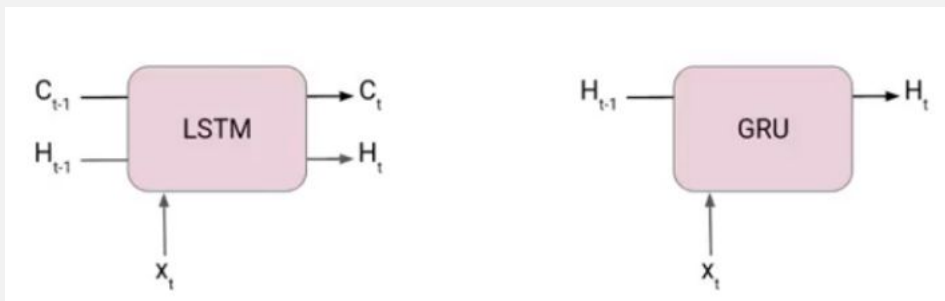
Long Short-Term Memory (LSTM) networks introduce gates i.e., input gate, output gate and forget gate that control the flow of information within the network, allowing them to learn long-term dependencies and it's popular choice for complex tasks.



Time series models

Gated Recurrent Unit (GRU)

Gated Recurrent Unit (GRU): Similar to LSTMs, GRUs use gates to manage information flow. However, they have a simpler architecture, making them faster to train while maintaining good performance. This makes them a good balance between complexity and efficiency.



[SOURCE](#)

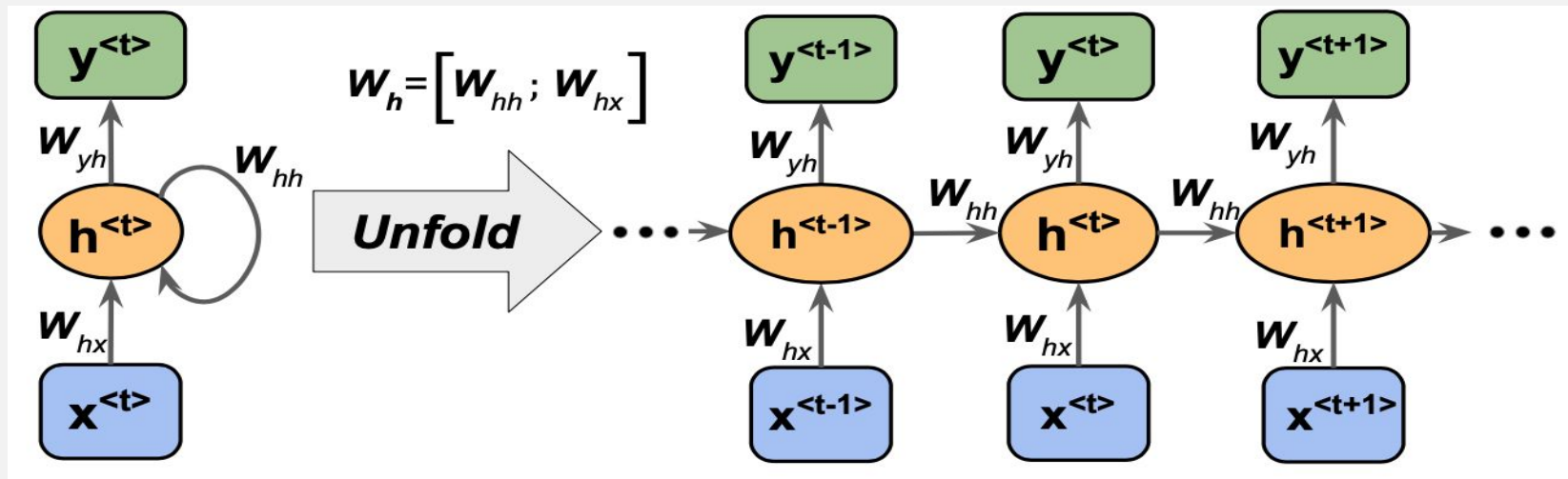


RNN's Architecture

► The Architecture of a Traditional RNN

The Architecture of a Traditional RNN

RNNs are a type of neural network that has hidden states and allows past outputs to be used as inputs. They usually go like this:

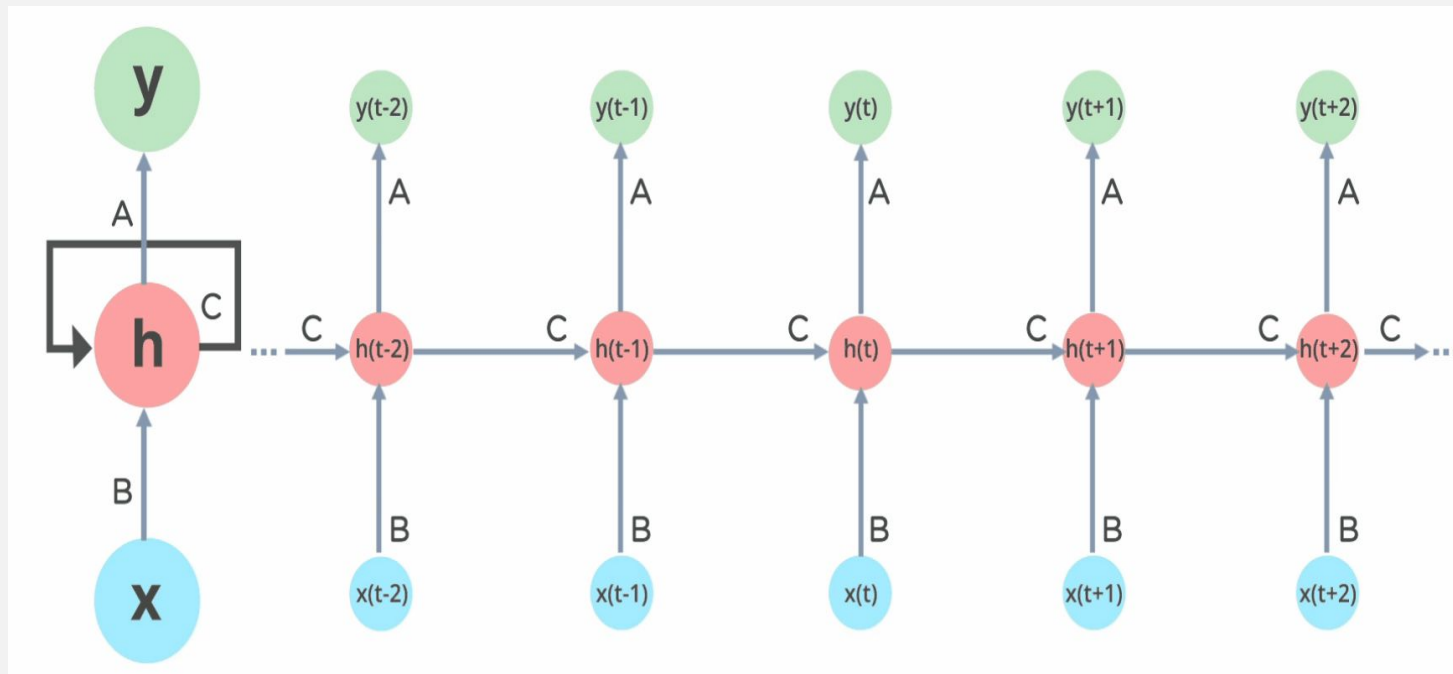


► How does Recurrent Neural Networks work?

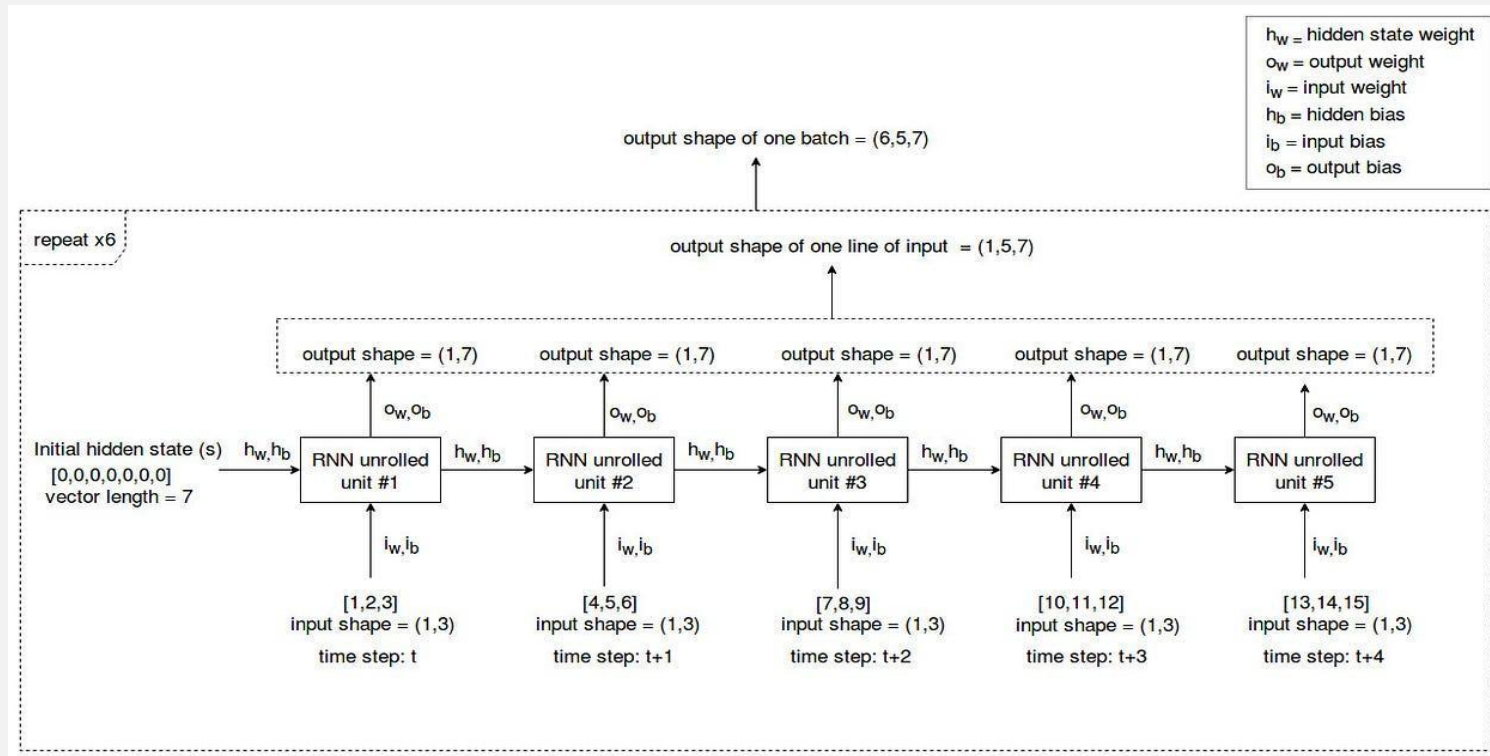
- The input layer x receives and processes the neural network's input before passing it on to the middle layer.
- In the middle layer h , multiple hidden layers can exist, each with its own activation functions, weights, and biases.
- RNN standardizes activation functions, weights, and biases across hidden layers, ensuring uniform characteristics.
- Instead of constructing numerous hidden layers, RNNs create only one and loop over it as needed, allowing for efficient processing of sequential data.



► How does Recurrent Neural Networks work?



► How does Recurrent Neural Networks work? (cont'd)





Recurrent Neural Network Training

Equations for Training:

1. Hidden State Calculation (h_t)

- (h_t) is computed using the previous hidden state (h_{t-1}) and the current input (x_t) followed by a non-linear activation function f (e.g., *tanh* or *sigmoid*).

$$h_t = f(W^{(hh)} h_{t-1} + W^{(hx)} x_t)$$

2. Output Prediction:

- (y_t) predicts the next element in the Sequential, typically using a SoftMax function to produce a probability distribution over the vocabulary.

$$y_t = \text{softmax}(W^s h_t)$$

3. Loss Calculation:

- (L_t) measures the error between the predicted and actual elements at time step t often using the cross-entropy loss function.

$$L^t(\theta) = \sum_{i=1}^{|V|} y_{ti} \log y_{ti}$$



► RNN's Architecture

Recurrent Connection is a key distinction of RNNs is the recurrent connection within the hidden layer. This connection allows the network to pass the hidden state information (the network's memory) to the next time step. It's like passing a baton in a relay race, carrying information about previous inputs forward.



Activation Function

Activation Function: This function introduces non-linearity into the network, enabling it to learn complex patterns. It transforms the combined input from the current input layer and the previous hidden layer state before passing it on.

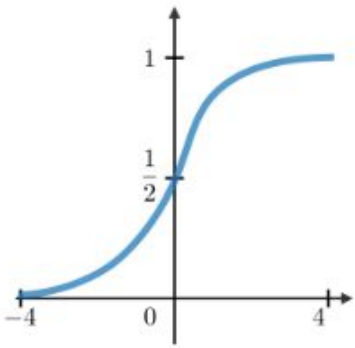
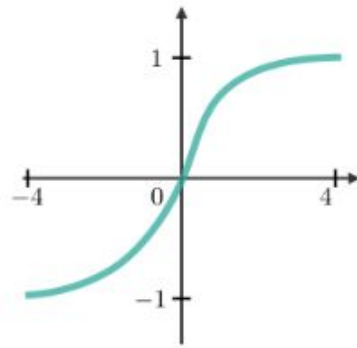
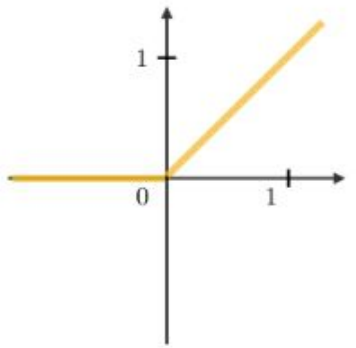
Common activation function : Sigmoid , ReLu , and TanH.

A neuron's activation function dictates whether it should be turned on or off. Nonlinear functions usually transform a neuron's output to a number between 0 and 1 or -1 and 1. The following are some of the most commonly utilized functions:



Common Activation Functions

Nonlinear functions usually transform a neuron's output to a number between 0 and 1 or -1 and 1.

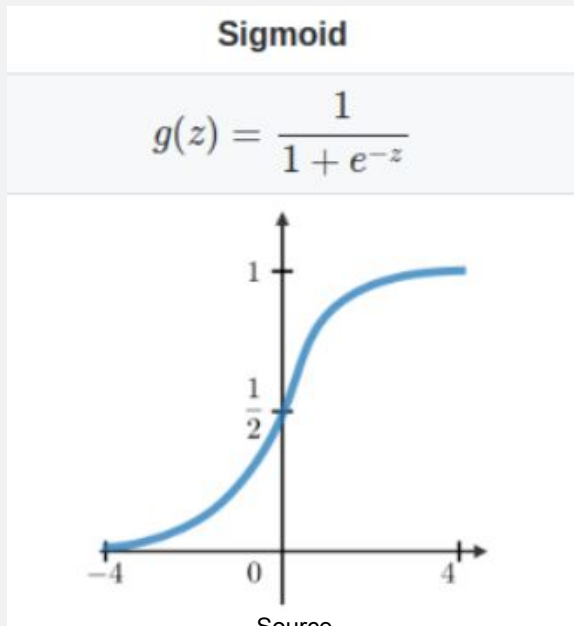
Sigmoid	Tanh	RELU
$g(z) = \frac{1}{1 + e^{-z}}$	$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	$g(z) = \max(0, z)$
		



► Activation Function

Sigmoid Function ($\sigma(x)$)

- Formula: $\sigma(x) = 1 / (1 + e^{(-x)})$
- Behaviour: Squishes any real number between 0 and 1.



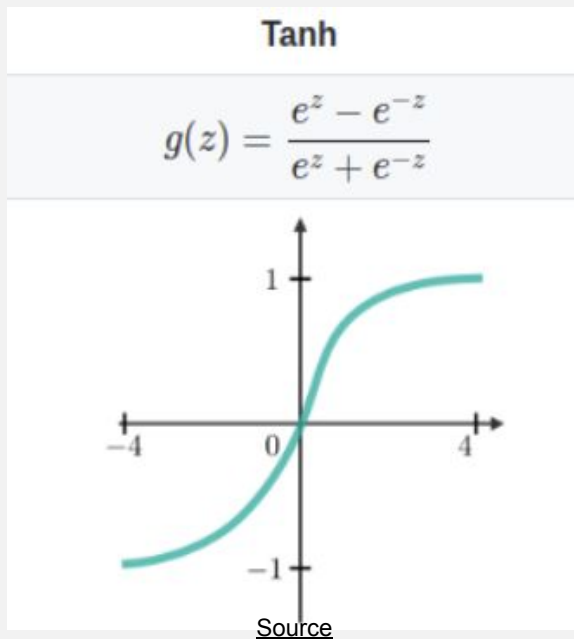
Source



Activation Function

Hyperbolic Tangent ($\tanh(x)$)

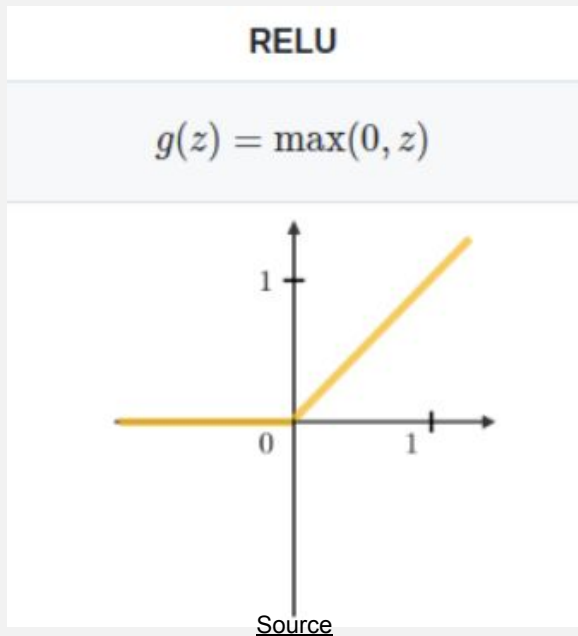
- Formula: $\tanh(x) = (e^x - e^{-x}) / (e^x + e^{-x})$
- Behaviour: Squeezes any real number between -1 and 1.



► Activation Function

Rectified Linear Unit (ReLU)(x)

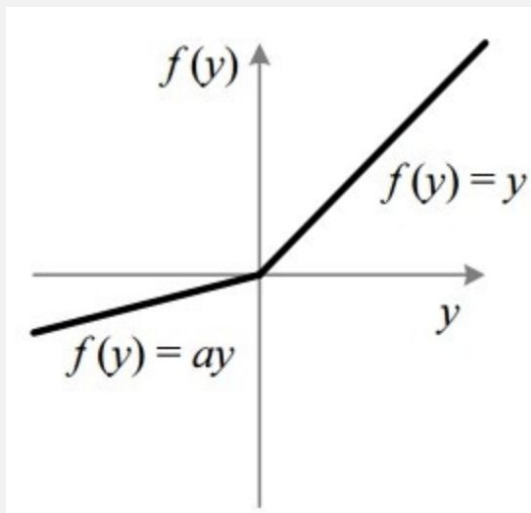
- Formula: $\text{ReLU}(x) = \max(0, x)$
- Behaviour: Outputs the input value if positive, otherwise outputs 0.



► Activation Function

Leaky ReLU (Leaky ReLU(x))

- Formula: $\text{Leaky ReLU}(x) = \max(\alpha * x, x)$ (where α is a small positive value, typically 0.01)
- Behavior: Similar to ReLU, but for negative inputs, it outputs a small fraction of the input instead of 0.



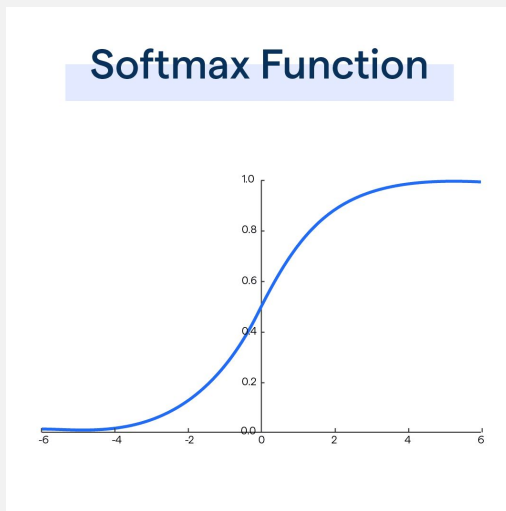
Source



► Activation Function

SoftMax (SoftMax(x))

- Formula: $\text{softmax}(x)_i = \exp(x_i) / \sum(\exp(x_j))$ (where i represents an element in the vector x and \sum denotes the sum over all elements j in x)
- Behavior: Converts a vector of real numbers into a probability distribution where all elements sum to 1.



Source

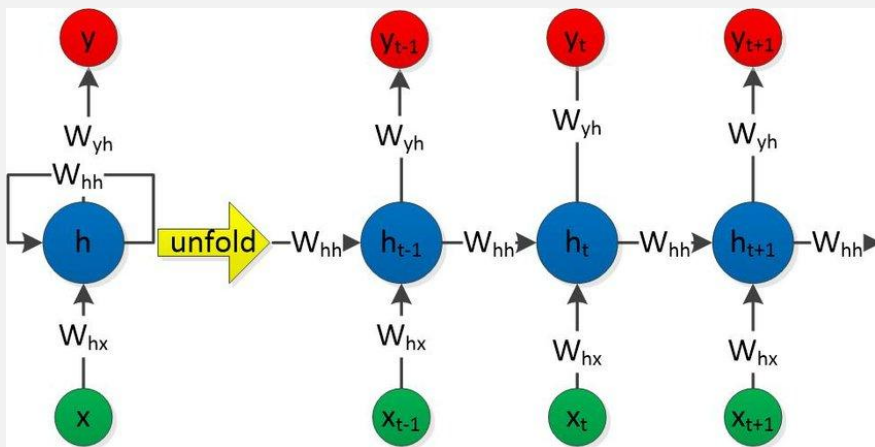


Mathematics behind RNN

To understand sequential models better, let's closely examine their key components.

Input Layer x_t :

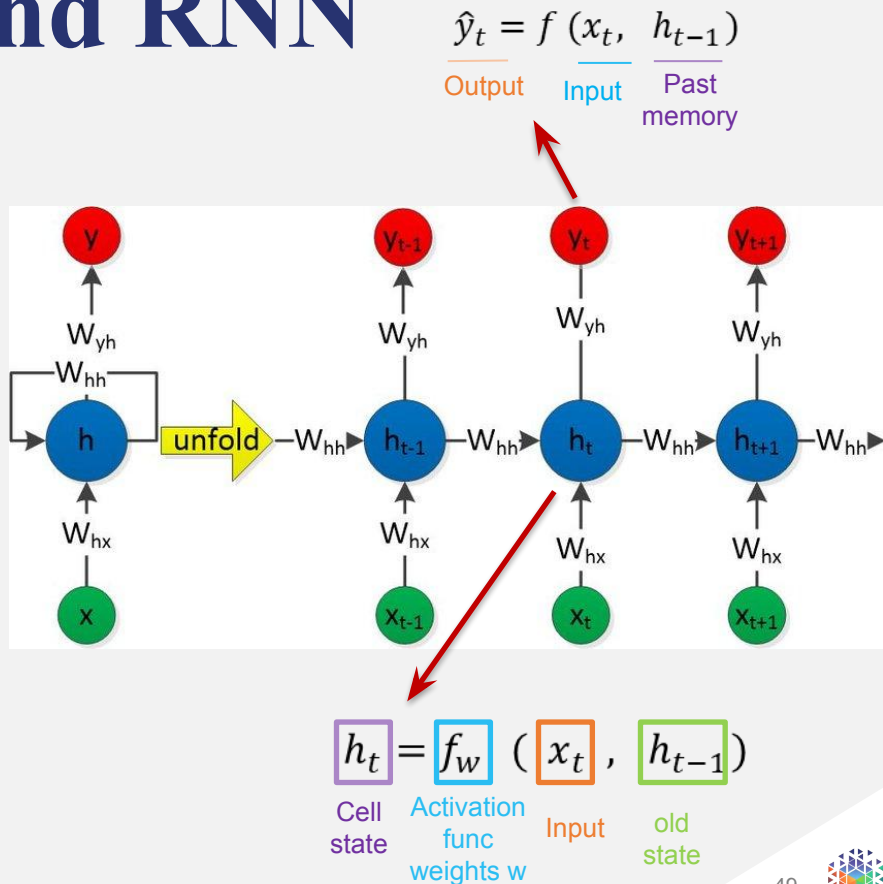
- Input vector It's a normal vector that consists of your inputs.



Mathematics behind RNN

Weights:

- $W^{(xh)}$: Weights that are used to transform input in a way that is consumable by the hidden state
- $W^{(hh)}$: Weights that define the relationship between the previous hidden state and the current hidden state
- $W^{(yh)}$: Weights that are used to transform the hidden state output to a prediction-based output.
- Same weight matrices and bias are shared across the RNN cells.

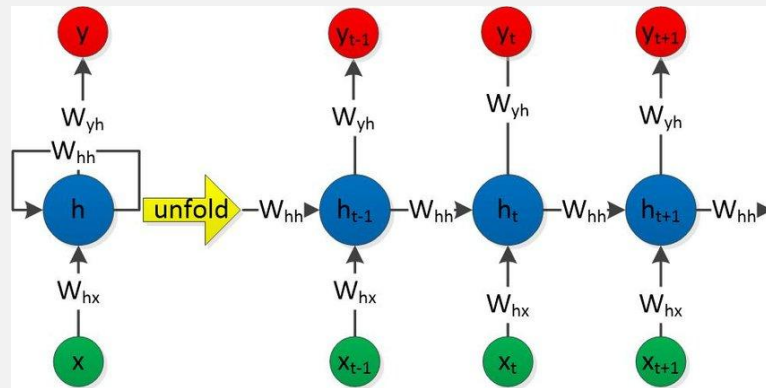


Mathematics behind RNN

Hidden State h_t :

- Acts as a relay of information in the RNN.
- Passes data from one neural network to the next.
- Captures context for predictions.
- Determined by current input x_t and previous hidden state h_{t-1} .
- Utilizes a nonlinear function like \tanh for computation.
- Maintains consistency across all time steps in the Sequential.

Note: The same function and set of parameters are used at every time step. RNNs have a state h_t that is updated at each time step as a Sequential is processed



$$h_t = \tanh (W^{(hh)} h_{t-1} + W^{(hx)} x_t)$$

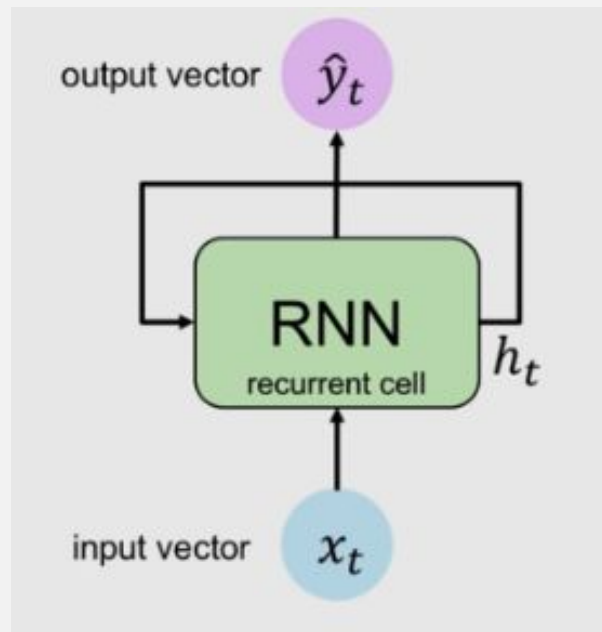


► Mathematics behind RNN

Output Vector y_t

- The output vector is nothing but uses the current hidden state h_t and $W^{(hy)}$ which basically helps transform the hidden state to output predicted values.

$$y_t = f(W^{hy} h_t)$$

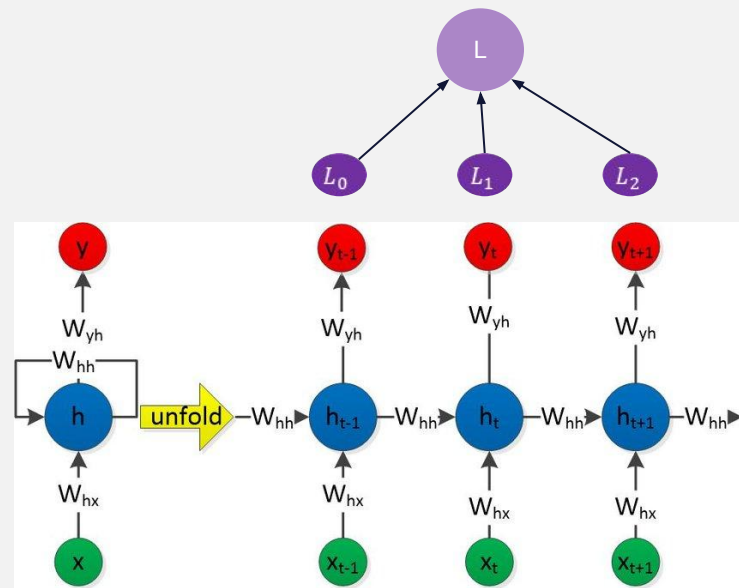


Mathematics behind RNN

Loss function

- The L_0 , L_1 , L_2 , etc. that you see in the diagram are the losses calculated at each stage and summed up to calculate L .
- RNN task is to minimize that final loss function using backpropagation

$$L = \sum_{t=1}^T L_t$$





Basic Python Implementation (RNN with Keras)

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

model = Sequential()
model.add(SimpleRNN(64, input_shape=(28,), activation='relu'))
model.add(Dense(units=dense_units, activation='linear'))


model.compile(loss='mean_squared_error', optimizer='adam')
```

This code initializes a Sequential model and adds a SimpleRNN layer followed by a Dense output layer.




Advantages and Disadvantages of RNN


PROS



Handle sequential data effectively, including text, speech, and time series.




Process inputs of any length, unlike feedforward neural networks.




Share weights across time steps, enhancing training efficiency.

CONS



Prone to vanishing and exploding gradient problems, hindering learning.



Training can be challenging, especially for long Sequentials.



Computationally slower than other neural network architectures.



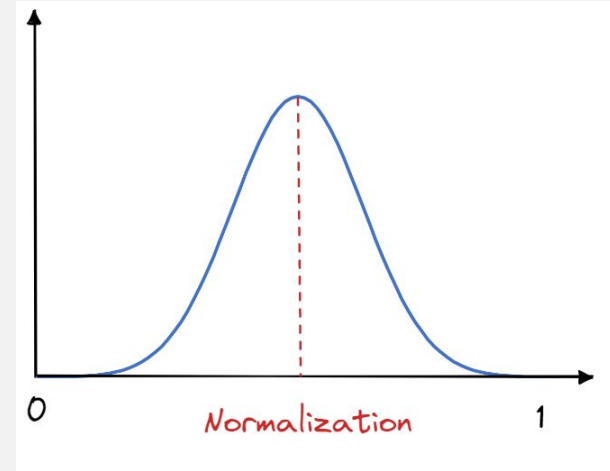
Preprocessing for Time Network Analysis

► Preprocessing for Time Network Analysis

Normalization

Normalization is a method used to organize data in a database. It involves scaling the data to reduce duplication, where values are adjusted to a range between 0 and 1.

When there are no outliers since it can't handle them, normalization is employed to remove the undesirable characteristics from the dataset.



[SOURCE](#)



► Preprocessing for Time Network Analysis

Normalization

In Python we use a transformer from the Scikit-Learn package called **MinMaxScaler** for Normalization.

$$x_{\text{norm}} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

[SOURCE](#)



► Preprocessing for Time Network Analysis

Normalization

In Python we use a transformer from the Scikit-Learn package called **MinMaxScaler** for Normalization.

```
from sklearn.preprocessing import MinMaxScaler
import numpy as np

data = np.array([[1, 2], [2, 4], [3, 6], [4, 8], [5, 10]])

scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(data)

print(scaled_data)
```



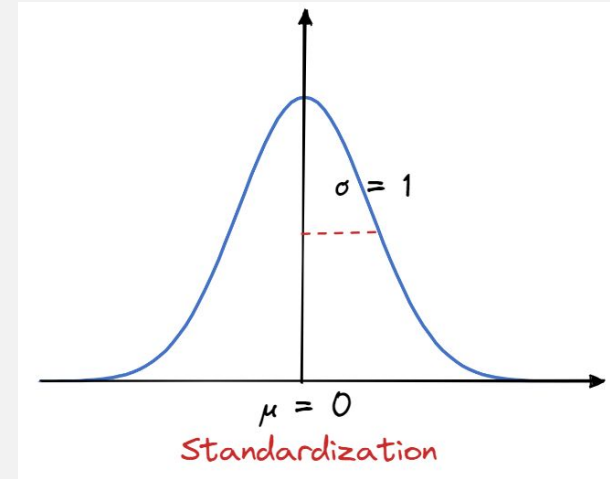
```
→ [[0.  0.  ]
   [0.25 0.25]
   [0.5  0.5  ]
   [0.75 0.75]
   [1.   1.   ]]
```



► Preprocessing for Time Network Analysis

Standardization

Standardization is the process of converting data into a common format to maintain consistency and compatibility across various systems and platforms. In data handling, this involves adjusting values so they have a mean of zero and a standard deviation of one. This approach is especially beneficial in machine learning and statistical analysis, as it enables data to be compared on the same scale, simplifying interpretation and analysis.



[SOURCE](#)



► Preprocessing for Time Network Analysis

Standardization

In Python we use a transformer from the Scikit-Learn package called **StandardScaler** for standardization.

$$x_{\text{stand}} = \frac{x - \text{mean}(x)}{\text{standard deviation}(x)}$$

[SOURCE](#)



► Preprocessing for Time Network Analysis

Standardization

In Python we use a transformer from the Scikit-Learn package called **StandardScaler** for standardization.

```
from sklearn.preprocessing import StandardScaler
import numpy as np

data = np.array([[1, 2], [2, 4], [3, 6], [4, 8], [5, 10]])

scaler = StandardScaler()
standardized_data = scaler.fit_transform(data)

print(standardized_data)
```



```
[[-1.41421356 -1.41421356]
 [-0.70710678 -0.70710678]
 [ 0.          0.         ]
 [ 0.70710678  0.70710678]
 [ 1.41421356  1.41421356]]
```



► Preprocessing for Time Network Analysis

Normalization vs Standardization

Standardization	Normalization
This method scales the model using the mean and standard deviation.	This method scales the model using minimum and maximum values.
When a variable's mean and standard deviation are both set to 0, it is beneficial	When features are on various scales, it is functional.
Values on a scale are not constrained to a particular range.	Values on the scale fall between $[0, 1]$ and $[-1, 1]$
This process is called Z-score normalization.	.Additionally known as scaling normalization
When the feature distribution is consistent, it is helpful.	.When the feature distribution is unclear, it is helpful



► Remove Non-Stationarity

Removing non-stationarity from time series data is essential for accurate forecasting because many forecasting models are based on the assumption that the statistical properties of the time series—such as mean, variance, and autocorrelation—remain constant over time. Non-stationarity in a time series can appear as trends, seasonality, or other irregular patterns. Addressing these issues helps ensure that the models produce reliable and meaningful forecasts.



► So, What is non-stationarity?

Non-stationarity refers to a characteristic of a time series in which its statistical properties, such as mean and variance, change over time. This means that the data series does not have consistent statistical characteristics across different time periods. Non-stationarity can manifest in several forms, including trends (systematic increases or decreases), seasonality (regular and predictable patterns that repeat over specific intervals), and other irregular patterns that can impact the analysis and forecasting of the time series.



▶ key components of non-stationarity

Trend: A trend is a long-term increase or decrease in the data over time. It can take various forms, such as linear or exponential, and reflects systematic changes in the series over a period.

Seasonality: Seasonality refers to periodic fluctuations or patterns in the data that occur at regular intervals. For example, retail sales might increase every year during the holiday season.



▶ key components of non-stationarity

Variance: Variance measures the dispersion or spread of data points around the mean. Non-constant variance, also known as heteroscedasticity, can indicate non-stationarity in a time series.

Autocorrelation: Autocorrelation occurs when the correlation between observations at different time points is not constant. A changing autocorrelation structure over time can also suggest non-stationarity in the data.



► How to remove non-stationarity?

► Trend:

Detrending:

- Fitting a Regression Line: Fit a regression line to the data to model the trend component and then subtract it from the original data, isolating the residuals.
- Moving Averages: Use moving averages to smooth the data, effectively removing fluctuations and leaving behind the underlying trend.

Differencing:

- Single Differencing: Subtract consecutive observations to remove the trend, resulting in a series of differences. This process can be repeated until the data appears stationary.
- Multiple Differencing: If a single differencing operation doesn't achieve stationarity, apply differencing multiple times. This involves taking the difference of differences until the desired stationarity is reached.



► How to remove non-stationarity?

Seasonality:

Seasonal Adjustment:

- **Seasonal Decomposition:** Apply techniques like Seasonal and Trend decomposition using Loess (STL) to separate the seasonal, trend, and residual components of the data. This method helps to identify and isolate the seasonal patterns from the overall series.

Seasonal Differencing:

- **Seasonal Differencing:** Subtract observations from the same season of different years (e.g., subtract the value of a particular month from the same month in the previous year) to remove the seasonal effect. This method can be repeated if the seasonal pattern persists after the first differencing.



► How to remove non-stationarity?

► Variance:

Transformation: Apply transformations such as logarithmic, square root, or Box-Cox transformation to stabilize the variance and make it more constant over time.

Autocorrelation:

Differencing: Besides removing trends, differencing can also help reduce autocorrelation by eliminating dependence between consecutive observations.

Autoregressive Integrated Moving Average (ARIMA): Utilize ARIMA models, which incorporate differencing to handle autocorrelation. (You'll study ARIMA in day 3 this week)



Evaluation Metrics

► Evaluation matrices

Evaluation metrics are quantitative measures used to assess the performance and effectiveness of a statistical or machine learning model. These metrics provide insights into how well the model is performing and help in comparing different models or algorithms. Such as :

- Mean Absolute Error (MAE)
- Mean Squared Error (MSE)
- Root Mean Squared Error (RMSE)
- Mean Absolute Percentage Error (MAPE)



► Evaluation matrices

Mean Absolute Error

The Mean Absolute Error (MAE) represents the average of the absolute differences between the actual and predicted values in a dataset. It measures the average magnitude of the residuals in the dataset.

$$MAE = \frac{1}{N} \sum_{i=1}^N |y_i - \hat{y}|$$

→ `sklearn.metrics.mean_absolute_error`

Where,

\hat{y} – predicted value of y

\bar{y} – mean value of y



► Evaluation matrices

Mean Squared Error

Mean Squared Error (MSE) represents the average of the squared differences between the actual and predicted values in a data set. It quantifies the variance of the residuals.

$$MSE = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y})^2$$

→ `sklearn.metrics.mean_squared_error`



► Evaluation matrices

Root Mean Squared Error

Root Mean Squared Error (RMSE) is the square root of the Mean Squared Error (MSE). It measures the standard deviation of the residuals.

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{N} \sum_{i=1}^N (y_i - \hat{y})^2}$$



```
from sklearn.metrics import mean_squared_error
```



► Evaluation matrices

Mean Absolute Percentage Error

Mean Absolute Percentage Error: this is the same as MAE but is computed as a percentage, which is very convenient when you want to explain the quality of the model to management,

$$MAPE = \frac{100}{n} \sum_{i=1}^n \frac{|y_i - \hat{y}_i|}{y_i}$$



RNN Applications

RNN Applications

Natural Language Processing (NLP)

RNNs can predict the next word in a sentence for text generation and autocomplete.



Time-Series Analysis and Forecasting

RNNs can predict stock prices, currency exchange rates, and other financial variables.



Music Generation

RNNs can learning patterns from existing music and producing new composition



Language Generation and Dialogue Systems

RNNs power chatbots by generating coherent responses in conversations.



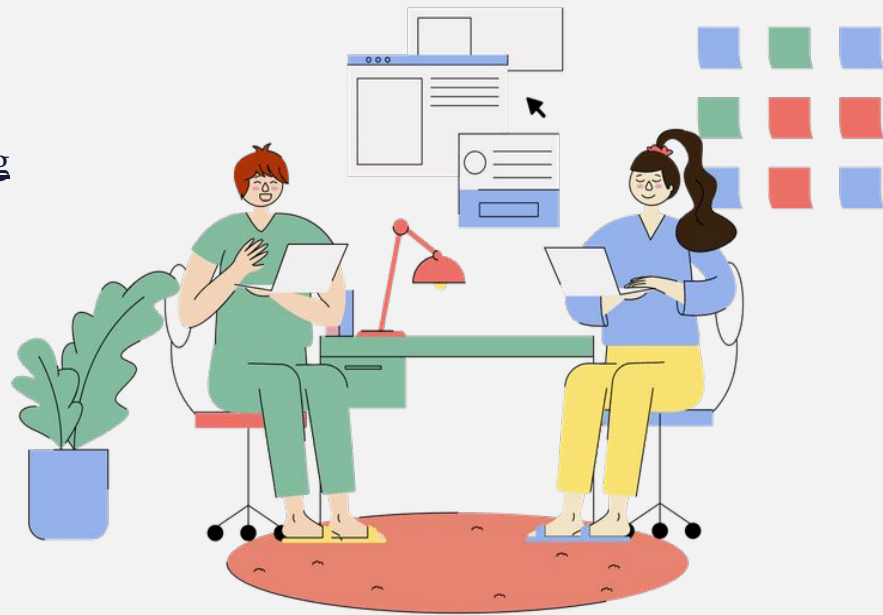
▶ Let's Practice

1st Tutorial:

- [5-Introduction to time series/2-Time Series Forecasting \(RNN\)/LAB/RNN_Tutorial.ipynb](#)

2nd Tutorial:

- [4-Deep learning/2-Time Series Forecasting \(RNN\)/LAB/RNN_Tutorial2.ipynb](#)



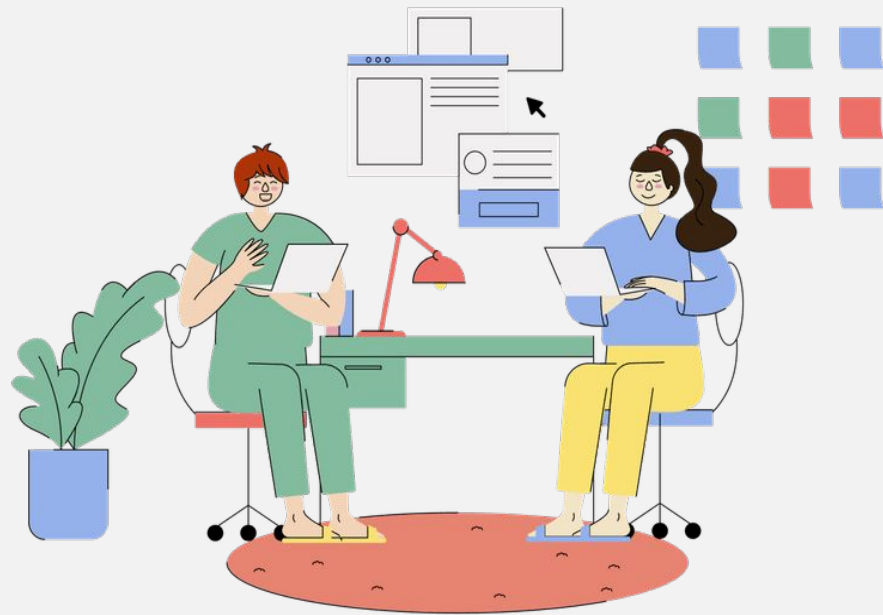
▶ Let's Practice

Exercise:

- [4-Deep learning/5-Time Series Forecasting \(RNN\)/LAB/RNN_Exercise.ipynb](#)

Dataset:

- [4-Deep learning/5-Time Series Forecasting \(RNN\)/LAB/Datasets/traffic.csv](#)



THANK YOU



SDAIA
الهيئة السعودية للبيانات
والذكاء الاصطناعي
Saudi Data & AI Authority