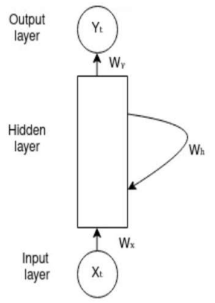
Limitations of Traditional Neural Networks

Limitations of traditional neural networks, such as FeedForward Neural Networks (FNNs), when it comes to processing sequential data. FNN takes inputs and process each input independently through a number of hidden layers without considering the order and context of other inputs. Due to which it is unable to handle sequential data effectively and capture the dependencies between inputs. As a result, FNNs are not well-suited for sequential processing tasks such as, language modeling, machine translation, speech recognition, time series analysis, and many other applications that requires sequential processing.

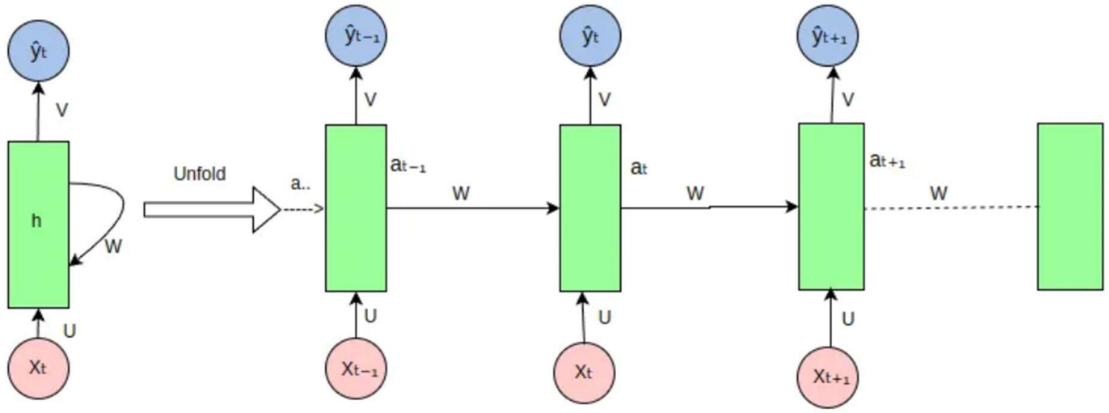
So, here RNN comes in frame.

RNN overcome these limitations by introducing a recurrent connection that allow information to flow from one time-step to the next. This recurrent connection enables RNNs to maintain internal memory, where the output of each step is fed back as an input to the next step, allowing the network to capture the information from previous steps and utilize it in the current step, enabling model to learn temporal dependencies and handle input of variable length.



Recurrent Neural Network

Recurrent Neural Network (RNN) is a type of Neural Network where the output from the previous step is fed as input to the current step. In traditional neural networks, all the inputs and outputs are independent of each other. Still, in cases when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words. Thus RNN came into existence, which solved this issue with the help of a Hidden Layer. The main and most important feature of RNN is its **Hidden state**, which remembers some information about a sequence. The state is also referred to as *Memory State* since it remembers the previous input to the network. It uses the same parameters for each input as it performs the same task on all the inputs or hidden layers to produce the output. This reduces the complexity of parameters, unlike other neural networks.



At each time step t, the hidden state a_t is computed based on the current input x_t , previous hidden state a_{t-1} and model parameters as illustrated by the following formula:

$a_t = f(a_{t-1}, x_t; \theta)$ ……..(1)

It can also be written as,

$a_t = f(U * X_t + W * a_{t-1} + b)$
where,

- a_t represents the output generated from the hidden layer at time step t .
- θ represents a set of learnable parameters(weights and biases).
- x_t is the input at time step t.
- U is the weight matrix governing the connections from the input to the hidden layer
- W is the weight matrix governing the connections from the hidden layer to itself (recurrent connections)
- V represents the weight associated with connection between hidden layer and output layer; $V \in \theta$
- a_{t-1} is the output from hidden layer at time t-1.
- b is the bias vector for the hidden layer
- f is the activation function.

Project:

Time Series Prediction with LSTM Recurrent Neural Networks in Python with Keras

Problem Statement:

Time series forecasting is a crucial aspect of business analytics, enabling companies to predict future trends based on historical data. Accurate forecasting can significantly impact decision-making processes, inventory management, and overall business strategy. One of the powerful tools for time series forecasting is Prophet, an open-source library developed by Facebook's Core Data Science team.

Steps for Implementing Prophet for Store Sales Forecasting

- Step 1: Import Necessary Libraries
- Step 2: Load the Dataset
- Step 3: Data Analysis
- Step 4: Preparing the Data for Prophet
- Step 5: Training the Prophet Model
- Step 6: Forecasting with Prophet
- Step 7: Plot the Forecast and the Component

Dataset

The dataset used collects daily historical sales data from 1,115 Rossmann stores over 31 months resulting into 1+ million transaction records. In addition to transactions, the dataset includes information about promotions, competition and holiday periods at store level. This info is used to incorporate holidays during forecasting step. The dataset is available on kaggle.

The Prophet Model

Trend Component: This component models the overall trend of the data, which can be linear or logistic. Prophet automatically detects changes in trends by selecting changepoints from the data.

Holiday Component: This component incorporates the impact of holidays on the data. Users can provide a list of important holidays to be included in the model.