Frontiers in Health Informatics
ISSN-Online: 2676-7104
www.healthinformaticsjournal.com
2024; Vol 13: Issue 2
Open Access

Enhancing COVID-19 Impact Prediction through the Application of LSTM Networks in Deep Learning.

Vinayak Ashok Bharadi[1], Bhushankumar Nemade[2], Sujata S. Alegavi[3], Ravita Mishra[4], Pravin Jangid[5], Namdeo Badhe[6],
[1] Finolex Academy of Management and Technology, Ratnagiri, Maharashtra, India
[2,5] Shree L R Tiwari College of Engineering, Mira Road, Thane, Maharashtra, India
[3,6] Thakur College of Engineering and Technology, Mumbai, India
[4]Vivekanand Education Society's Institute of Technology, Mumbai, India.
Corresponding author. E-mail: bnemade@gmail.com

Article Info

Article type:

Research

Article History:

Received: 2024-03-19

Revised: 2024-05-10

Accepted: 2024-06-22

Keywords:

COVID-19, LSTM, ANN, Neural Network, Sequence Prediction, Deep Learning, Keras.

A B S T R A C T

Artificial Neural Networks (ANNs) have been around for a while, and as technology has progressed, more people can have now access to Graphical Processing Units (GPUs), Tensor Processing Units (TPUs), and complex architectures. These days, deep neural networks are of the utmost significance in pattern recognition. One special application of the ANNs is the sequence classification and prediction. A special type of neural network with the capacity to remember patterns along with the temporal aspects have been widely used, they are the recurrent Neural Networks (RNNs). The Long Short Term Memory Networks (LSTMs) are improved versions of RNN with a better dealing of vanishing gradient problems. In this chapter, we discuss an LSTMs with their regular implementation as well as time distributed and bidirectional implementations for the purpose of sequence prediction. Every day, new information about the COVID-19 pandemic's effects is released, and people all over the world are still dealing with its aftermath. Long Short-Term Memory (LSTM) networks are trained with this data in order to predict estimates of the global impact of the COVID-19 pandemic. The LSTM architectures are discussed and compared with a vanilla RNN and the results are presented here. The results show the LSTMs outperform RNNs when the mean absolute error is compared for all the models.

1. INTRODUCTION

An period of unparalleled challenges and worldwide uncertainty has been brought about by the COVID-19 pandemic, that started in late 2019 with the advent of the newly discovered coronavirus SARS-CoV-2. This viral outbreak has disrupted every facet of human existence, affecting public health, economies, and daily life on an unprecedented scale. In the face of these daunting complexities, the imperative to predict and understand the consequences of the virus has never been more urgent. One of the most promising tools for addressing this challenge is the realm of deep learning, particularly the application of LSTM networks. In the past decade, Neural Networks (NN), Deep Learning (DL), and Artificial Intelligence (AI) have become ubiquitous, driving innovations across a spectrum of technological domains. These advancements have left an perpetual mark on fields like a signal processing, pattern recognition, image classification, and computer vision, with their tangible impacts exemplified in everyday virtual assistants like Apple Siri, Google Assistant, and Microsoft Cortana. These intelligent companions rely on signal processing (SP), speech recognition (SR), natural language processing (NLP), and DL, all underpinned by ANN—the foundational pillars.

Recent years have witnessed the ascension of NN, and DL to the forefront of technological innovation. Their accomplishments have been highlighted not only within the scientific community but also in mainstream media. The proliferation of open-source libraries like TensorFlow, PyTorch, Keras, and Flux.jl has democratized access to

these technologies, fostering their widespread adoption and adaptability. Within the context of this research, several methodologies deserve our attention alongside LSTM networks. The Neural Network referred in the context of current work are also called ANN. ANNs are massively parallel systems having a huge number of interconnected neurons or simple processors implemented as computational algorithms. They come under a category of information processing system, working in the same way as the human brain processes information. Deep learning (DL) is a particular technique of machine learning that integrates the artificial neural networks in successive layers for the iterative learning from the input data. DL gives you the edge when you need to learn patterns from unstructured data. DL implementations has the computational nodes created and networked, with each node specializing in a particular aspect of the learning process, replicating how humans' read, analyze, and decode information and accordingly make the decisions [12].

DL enables computers to be trained to handle complex situations and unclear abstractions. Deep learning networks (DLN) are extensively utilized for computer vision tasks, speech recognition, and image and video analytics and processing. Deep learning is the new edge in automated decision making. ML and DL offers potential value to the applications designed for big data analytics and help them to find the insights by better understanding of the subtle changes. It is apparent that many events in companies and sectors are too complex to be fully understood by a single question. Rather, an advanced solution that can reveal hidden patterns and predict future events is needed [2][13]. In this experimentation, application of a type of DLN called as LSTM for the problem of COVID 19 [3][14] related forecast. This is an application of the DLN for the regression analysis.

ANNs have been a part of our technology landscape for a long time, dating back to the 1940s and 1960s. This era saw the introduction of biological learning [1], as well as the deployment of the first ANN models, most notably the perceptron by Rosenblatt in 1958[3], this was focused on the training of a single neuron. The further developments were having connectionist approach of the 1980–1995 era, It gave us the back-propagation algorithm for training a NN with one or two hidden layers [4]. The current phase of development in ANN is also referred as third phase. This phase is focused on a key concept called as deep learning (DL), it started around 2006 [7][8][9], as the hardware capable of supporting deep Learning are in reach of common people, this phase is moving beyond the theory towards practical implementations and realizable applications.

Traditional time series forecasting methods, such as SARIMA, and ARIMA have long served as the bedrock of time-dependent data analysis. A comparative examination of these conventional approaches vis-à-vis the deep learning methods elucidated in this research promises invaluable insights. RNNs, a specialized subclass of artificial neural networks, are tailor-made for processing sequential data. In the present research, RNNs and their sophisticated variants play a pivotal role in capturing the temporal dependencies embedded within the COVID-19 data, endowing them with the power of prediction. Advanced adaptations of traditional RNNs, including Bidirectional LSTMs and Time Distributed LSTMs, enrich the network's ability to analyze sequential data by assimilating information from both past and future time steps. These extensions serve as indispensable components within the research's deep learning framework. Time Distributed LSTMs: Time Distributed LSTMs represent a significant stride in the evolution of the LSTM architecture, meticulously engineered to process data sequences over time. Their invaluable contribution comes to the fore when working with data structured in a temporal format. In this research, Time Distributed LSTMs significantly enhance the model's proficiency in generating predictions based on time series data.

Bidirectional LSTMs, as their name implies, operate bidirectionally when processing data sequences. This dual perspective empowers them to capture context from both preceding and succeeding time steps concurrently, making them pivotal in decoding the intricate patterns embedded within COVID-19 data. The introductory section of this research paper serves to establish the context and provide an expansive backdrop for the application of LSTM networks in forecasting the consequences of the COVID-19 pandemic. The study embarks on a journey to explore the evolution of deep learning techniques, anchored by LSTM networks, and how they stand in comparison to more traditional time series analysis methodologies like SARIMA ARIMA. It underscores the immense potential encapsulated within these advanced methodologies for addressing the unique challenges posed by the pandemic. The COVID-19 pandemic has underscored the need for innovative approaches to fathom and mitigate its repercussions. LSTM networks, as demonstrated in this research, proffer a potent tool for forecasting the consequences of the virus's propagation. By fusing data-driven insights with the potency of deep learning, we can make more enlightened decisions, allocate resources more judiciously, and ultimately strive toward a safer, more resilient future. This research significantly contributes to the burgeoning body of knowledge concerning the application of DL in epidemiology and public healthcare. It is our fervent hope that this endeavor will ignite further

exploration and innovation within this pivotal domain, serving as a beacon of light amid these challenging times.

2. COMPARATIVE ANALYSIS OF VARIOUS MODELS FOR TIME SERIES FORECASTING

In today's data-driven world, effective predictive modeling is essential for informed decision-making. Time series forecasting is a critical component of predictive analytics, enabling the understanding and anticipation of trends from historical data. This comparative study explores the strengths, weaknesses, and real-world applications of diverse time series forecasting models, including ARIMA, ANN, RNN, LSTM networks, and their extensions. By examining practicality, prediction accuracy, and computational aspects, it provides valuable insights for researchers, practitioners, and decision-makers in various domains.

2.1 Recurrent Neural Networks:

This type of NN processes sequential data, while processing the sequential values the output from preceding steps are provided as a input to the current step. In a conventional NN, all inputs and outputs do not dependent on one another, and inputs are supposed to be fed all at a time as a pattern or a vector for a classification type of a problems. However, while predicting the following word of a sentence, the prior words are necessary, and hence the previous words must be recalled [15]. This resulted in the invention of RNNs, that addressed the issue by include a Hidden Layer. The Hidden state, which is responsible for maintaining temporal information inside a sequence, is crucial to RNNs.

Feedforward Networks and its limitation

Before getting into the details of LSTMs, and RNNs, it's a good idea to first learn about feedforward networks. These networks are named after their method of routing information via a series of mathematical procedures executed at the network's nodes. Information flows straight without returning to a single node in feedforward networks, but RNNs and LSTMs cycle it through a loop, earning them the label "recurrent." This recurrent behavior is demonstrated by LSTMs, a specialized kind of RNN [16].

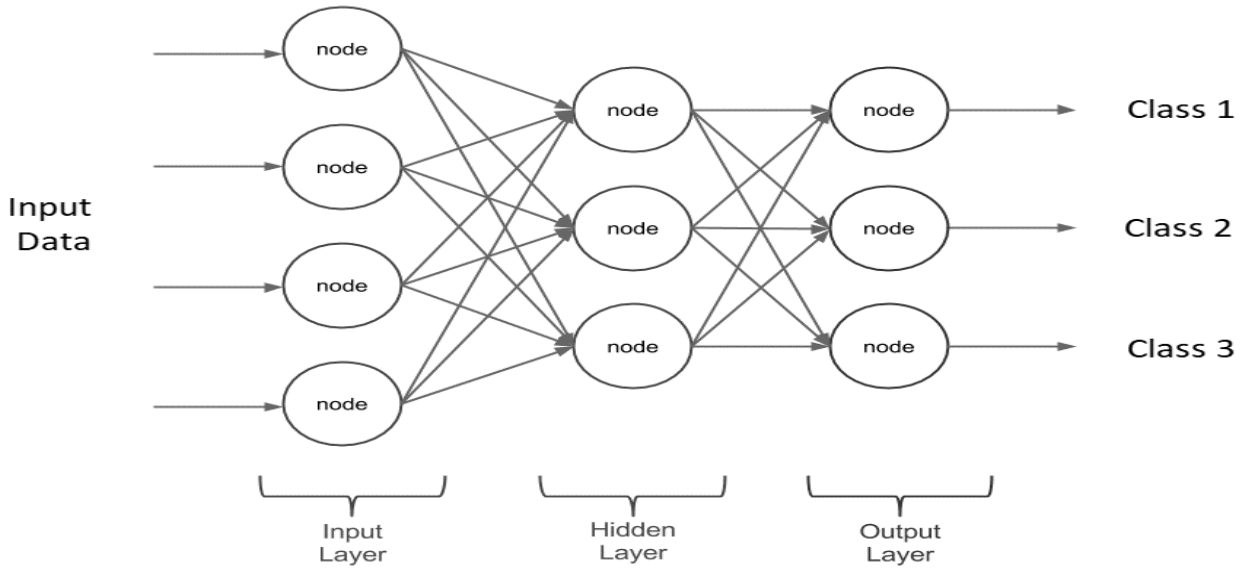

Fig. 1. Typical Feedforward Neural Network

The feedforward networks take the input patterns and produces a corresponding output indicating the category of the input, this will be a tag or class name in case of a classification problem with supervised learning. In this way the input data is classified into categories, an example can be input image is classified as an object as bus, car or human being. This is described in Fig 1, where we can see the input data value is classified in three classes.

A feedforward network undergoes training on labeled data until the classification error is minimized. The training process gives a set of parameters consisting of weights and biases, collectively this is called as a model. The trained model can further be exposed to the testing data such as a set of unseen photographs, other than the ones used for the training. The prediction for one set of photos will not affect the next predictions as all the predictions are independent of each other and that is quite natural.

In effect a feedforward network has no notion of order in time, or there is no temporal relation between the predictions. The only input it considers is the current data. This becomes a limitation when the current prediction depends on past data, one such example can be the weather forecasting. This has served as the impetus behind the design of RNNs.

For traditional neural networks, each input and output is unrelated to the others. They lack of temporal memory of the input sequential data. The RNNs addressed this limitation with the help of a Hidden Layers (HL) as depicted in figure 2. The fundamental and primary characteristic of RNN is the Hidden state, that retains important data of a sequence to a certain extent [17].

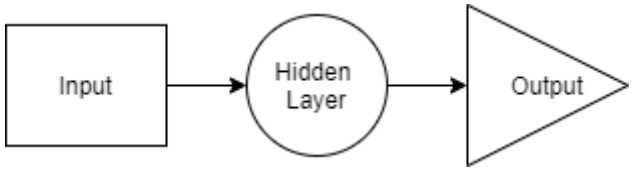

Fig. 2. Typical Neural Network Flow Diagram

Recurrent Neural Networks (RNNs) have a memory notion that allows them to save information from past calculations. They generate the output by using a consistent set of parameters for each input and conducting the same operation across all inputs or hidden layers. This reduces parameter complexity when compared to other forms of neural networks.

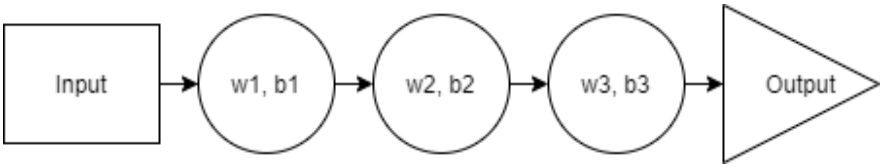

Fig. 3. Typical Neural Network Flow Diagram

As shown in Fig. 3, consider a neural network with a single input, one output, and three hidden layers (HL). The network is made up of three independent HLs, each one having their own set of biases and weights (w1, b1), (w2, b2), and (w3, b3). This represents each layer's independence, showing that they do not retain knowledge from prior outputs. It's a memoryless system.

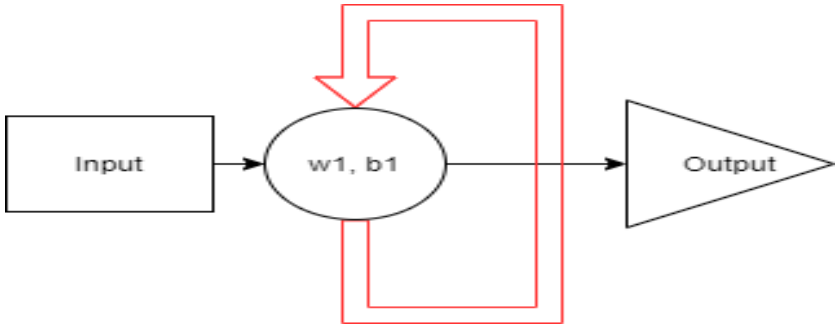

Fig. 4 RNN with Input from Previous step

If the data has to be processed by the RNN then it will perform the following steps: RNN gives all layers the same weights and biases, changing independent activations into interdependent activations. This operation further reduces the complexity of increasing parameters. The output of current time instance is given back to same layer. It

allows the RNN to retain past output by feeding it into the next hidden layer. Furthermore, the three HL are merged into a single recurrent layer with the same weights and bias as the other hidden layers.

$a_t = f(a_{(t-1)}, C_t)$ (1)

Where, h_t : present state , h_{t-1} : preceding state, Xt : input state

The equation 2 describes activation function,

$$A_t = \tanh(B_h h A_{t-1} + B_x h C_t) \quad (2)$$

Where, $B_h h$: *weight at recurrent neuron, and Bxh Weight at input neuron*

And the output is given by:

$y_t = W_{hy}\, h_t$ (3)

Where: y_t : output, w_{hy} : weight at output layer

2.1.1 Backpropagation Through Time (BPTT)

RNNs' goal is to classify sequential input while taking time into account. An RNN's training procedure is similar to that of a typical Artificial Neural Network (ANN). There is a modest adjustment to the backpropagation method. Because the parameters are shared across all time steps in the network, a gradient at each output is influenced not just by the present time step, but also by the time steps that preceded it[18].

Backpropagation in feedforward networks travels backwards from the ultimate mistake in the output, adjusting inputs, weights, and biases for every hidden layer in the network. These weights are modified based on their share of the error, which is determined using partial derivative(E/w). These partial derivatives are then employed in the learning process to adjust the weights in the direction of reducing error, similar to gradient descent.

RNNs use a backpropagation extension known as backpropagation over time (BPTT). In this context, time is defined as a clearly defined ordered collection of computations that link one time step to the next, comprising all backpropagation processes inside the execution cycle. Whether recurrent or not, artificial neural networks (ANNs) are fundamentally layered complex functions like f(g(h(x))). The addition of a time factor simply broadens the set of functions whose derivatives are determined using the chain rule.

Truncated BPTT is a related idea that serves as an approximation of complete BPTT, specifically tailored for extended sequences. Over multiple time steps, the computing cost per parameter update of the entire BPTT increases. The gradient can only flow back so far due to the truncation, restricting the network's ability to learn dependencies as far as full BPTT [19].

2.1.2 Vanishing (and Exploding) Gradients

RNNs have been a part of our technology landscape for a long time. However, during the 1990s, the vanishing gradient problem posed a substantial obstacle to recurrent net performance. The error gradient represents the change in all weights as a result of the error change. When the gradient is insufficiently significant, the weights cannot be modified in a direction that lowers error, resulting in learning difficulties and the learning process being halted. Recurrent nets found instability when attempting to connect a final output to events that occurred many time steps earlier. Determining the right priority to allocate to distant inputs is difficult since information going through neural nets is multiplied numerous times, leading gradients to become very small or explode depending on factors such as learning rate and network circumstances. Vanishing or inflating gradients are a prevalent problem in DLN, which connect layers and time steps by multiplication. Because explosive gradients can be trimmed or compressed, dealing with them is quite simple. However, vanishing gradients offer a more complex difficulty since they become small enough for computers to manage or for networks to learn properly. LSTM networks were created around the mid-1990s to address the problem of disappearing gradients [20].

2.1.3 RNN Extensions

RNNs have advanced in the form of complex variants designed to address limitations in the vanilla or traditional RNN model. Some notable versions are listed below. Bidirectional RNNs work on the basis that the outcome at time 't' is influenced by both past and future elements in the sequence. For example, when forecasting a missing number in a series, it is critical to account both past and future context. Bidirectional RNNs are simple: they are made up of two RNNs layered on top of one another. As shown in Figure 5, the output is calculated depending on the hidden state(HS) for both RNNs.

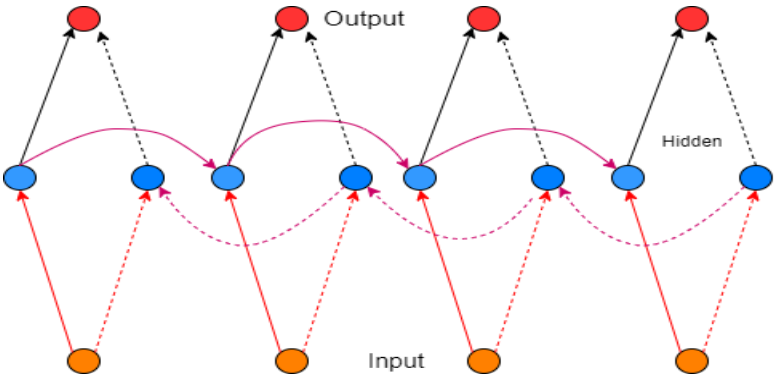

Fig. 5. Conceptual Diagram for Bidirectional RNN

Deep (Bidirectional) RNNs are similar to Bidirectional RNNs, only that these RNNs have multiple layers per time step. Addition of deep layers gives higher learning capacity provided lots of training data is there and you have to train a set of hyperparameters. Figure 6 shows the Conceptual Diagram for Deep Bidirectional RNNs [21][33][34].

These days, LSTM networks are fairly common. Despite the exception of the concealed state, they inherit the precise design of ordinary RNNs. The prior state and the present input are used as inputs to the memory of LSTMs (called cells). These cells internally select what to maintain and what to delete from memory [22][35]. The prior state, current memory, and input are then combined. This method addresses the vanishing gradient issue quickly.

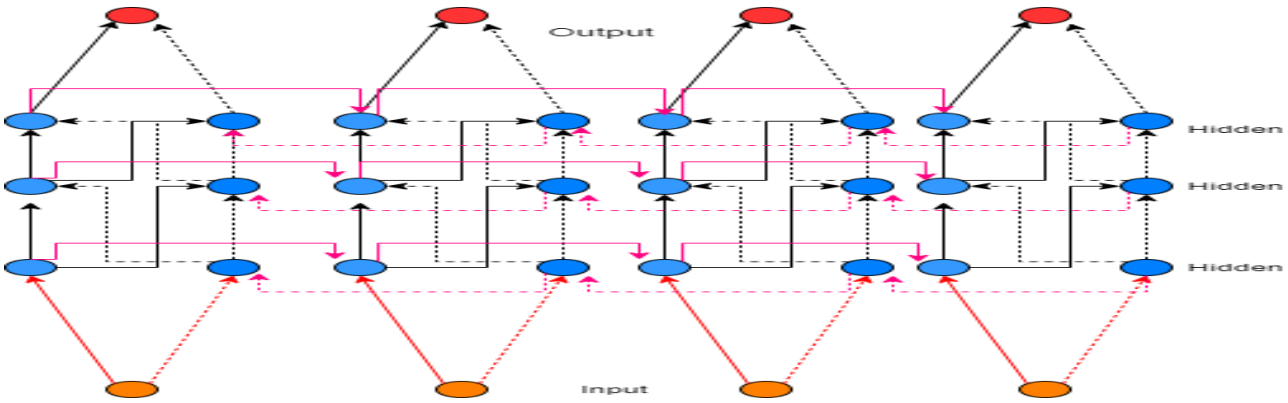

Fig. 6. Conceptual Diagram for Deep Bidirectional RNN

While Deep (Bidirectional) RNNs and Bidirectional RNNs are comparable, they are not the same since they contain many layers at each time step. Deep layers improve learning capacity, assuming adequate training data and the requirement to optimize a set of hyperparameters. Figure 6 depicts the Deep Bidirectional RNN Conceptual Diagram [21][41]. LSTMs have grown in prominence in recent years. With the noteworthy exception of the concealed state, they retain the underlying design of ordinary RNNs. Memory (referred to as cells) in LSTMs absorbs input from both the prior and present states. These cells make the internal decision about what to retain in memory and what to remove [22][36]. They then integrate the past state, present memory, and input, solving the vanishing gradient problem.

2.2 LSTM Networks

In the mid-1990s, German researchers Sepp Hochreiter and Juergen Schmid Huber introduced a recurrent nett version that included LSTMs. LSTM network were created to tackle the vanishing gradient problem and remember information for longer periods of time than standard RNNs. Because LSTMs are able to maintain a constant error, they may learn across several time steps and backpropagate across layers and time. By keeping a more consistent error, LSTMs allow RNNs keep learning across huge time steps (over 1000), enabling a channel to remotely link the causes and consequences. Since artificial intelligence and machine learning algorithms frequently operate in environments with weak, delayed, and sparse forwarding signals, this is one of the most challenging problems for them [25][38][39]. The equations in table 1 explain the most popular version of LSTM, and the matching LSTM cell is depicted in Figure 7.

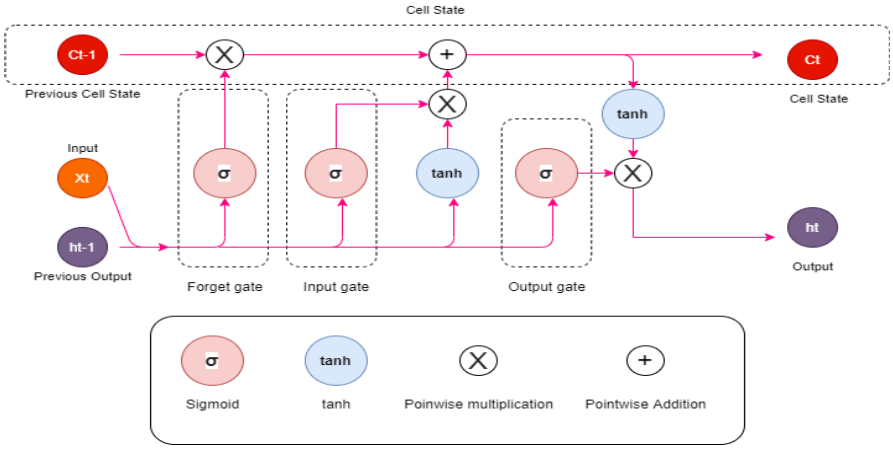

Fig. 7. Conceptual Diagram for Long Short-Term Memory Network Cell

Gated cells are used by LSTMs to retain information after the RNN's normal development. These cells give the network the ability to store and retrieve data inside of them, among other data manipulation capabilities. Cells are able to make decisions on their own based on information, and they can act by opening or closing gates. LSTM has an edge over standard RNNs in these types of tasks due to its long-term memory retention. Conceptually, LSTM is often depicted as a memory cell (cell state), denoted as ct, that may be subjected to write, read, and reset operations based on the forget gate (ft), input gate (it), and output gate (ot). The main distinction between these gates and basic RNNs is that the former utilise the sigmoid activation function, which restricts outputs to values between 0 and 1, while the latter use the tanh activation function. All of these gates change over time. These gates therefore have components that may be interpreted as ranging from completely off (0), which prevents information flow, to totally on (1), which permits unrestricted information flow.

Table 1 Gates in a conventional LSTM and their functions

Type of entity	Role	Equation
Input gate	Responsible for adding information to the cells.	$i_t = \sigma(W_{ih}h_{t-1} + W_{ix}x_t + b_i)$
Forget gate	Removes information that is no longer necessary for the completion of the task. This step is essential to optimizing the performance of the network.	$f_t = \sigma(W_{fh}h_{t-1} + W_{fx}x_t + b_f)$
Output gate	Determines what the next hidden state should be. Selects and outputs necessary information	$o_t = \sigma(W_{oh}h_{t-1} + W_{ox}x_t + b_o)$
Cell State	State of the LSTM Cell	$i_t = \sigma(W_{ih}h_{t-1} + W_{ix}x_t + b_i)$ $c_t' = \tanh(W_{ch}h_{t-1} + W_{cx}x_t + b_c)$ $c_t = f_t \odot c_{t-1} + i_t \odot c_t'$
Output	Output of LSTM	$h_t = o_t \odot \tanh(c_t)$

The components of the preceding cell state, c_{t-1}, combine with the candidate update, c_t, based on the forget gate, f_t, and the input gate, i_t. Further, the new hidden state h_t is generated by applying the memory cell with the final activation function and then the elements are weighting according to the output gate o_t. The vanishing gradient problem is addressed by the formation of the cell state which acts as a memory cell. The forget gate modulates the path between previous and current cell state (c_{t-1} and c_t), the forget gate decides what information to remembers and what to forget hence giving a notion of LSTM to the LSTM cell. Here is an example using python pseudo code-

```
def LSTM (prev_ct, prev_ht, prev_ht, input):
    addition = prev_ht+ input
    ft = forget_layer (addition)
    candidate_state = candidate_state_layer(addition)
    it = input_layer (addition)
    ct = prev_ct * ft + candidate_state * it
     ot = output_layer(addition)
    ht = ot * tanh(ct)
 return ht, ct
      ct = [0,0,0]
      ht = [0,0,0]
for input in inputs:
  ct, ht = LSTM (ct, ht, input)
```

In a nutshell the control flow of an LSTM network consists of a limited number of tensor operation plus a for loop. Predictions can be made using the concealed states. The LSTM can identify the importance of information to keep or eliminate during sequence processing by incorporating these techniques. In the ever-evolving landscape of data analysis and prediction, a spectrum of machine learning techniques and models plays a critical role. These include ANNs, SARIMA/ARIMA models, RNNs and their extensions, Time Distributed LSTMs, and Bidirectional LSTMs. Each of these models offers a unique set of characteristics, encompassing factors like training time, prediction accuracy, interpretability, complexity, scalability, memory efficiency, parallel processing capabilities, scoring metrics, hyperparameter tuning requirements, model selection methods, and real-world applications. Understanding these distinctions is essential for making informed decisions when selecting the most appropriate tool for addressing specific data-related challenges efficiently and effectively. The following Table 2 describes the comparative study of various methods.

Table 2: Comparative Investigation of diverse methods for Data Prediction and Analysis

Artificial Neural Networks (ANN)	**SARIMA/ARIMA**	**Recurrent Neural Networks**	**RNN Extensions**	**Time Distributed LSTMs**	**Bidirectional LSTMs**
Training Time	Depends on data	Long	Short	Moderate	Moderate
Prediction Accuracy	Moderate to High	Moderate to High	High	Moderate to High	Moderate to High
Interpretability	High	Low	Moderate	Low	Low
Complexity	Low to High	Low to High	Low to High	Moderate to High	Moderate to High
Scalability	Scalable	Scalable	Limited	Scalable	Scalable
Benchmarking Tools	Various tools	TensorFlow, Keras, PyTorch, etc.	R's forecast	TensorFlow, Keras, PyTorch, etc.	TensorFlow, Keras, PyTorch, etc.
Memory Efficiency	-	Moderate	High	Moderate	Moderate
Parallel Processing	Supported	Supported	Not Supported	Supported	Supported
Scoring Metrics	ROC AUC, F1-score, RMSE, etc.	Accuracy, Loss, ROC AUC, F1-score, RMSE, etc.	AIC, BIC, RMSE, etc.	Accuracy, Loss, ROC AUC, F1-score, RMSE, etc.	Accuracy, Loss, ROC AUC, F1-score,etc.
Hyperparameter Tuning	Required	Required	Limited	Required	Required

Model Selection	Cross-validation, Grid Search, etc.	Cross-validation, Grid Search, etc.	Information Criterion	Cross-validation, Grid Search, etc.	Cross-validation, Grid Search, etc.
Real-world Applications	Fraud Detection, Recommender Systems, Sentiment Analysis, etc.	Image recognition, Natural Language Processing, Time Series Analysis, etc.	Time Series Forecasting, Anomaly Detection, etc.	Natural Language Processing, Speech Recognition, Language Translation, etc. Speech Recognition, Sequence Generation, etc.	Sequence Generation, Handwriting Recognition, Anomaly Detection, etc.

3. COVID DISEASE FORECASTING METHODOLOGY

A viral virus captured global interest in early 2020 due to its extraordinary speed of transmission and infections. This virus is considered to have started in a food market in Wuhan, China, in December 2019, and has since spread to countries as far away as the United States, the United Kingdom, and the Gulf countries [27][40]. Officially known as SARS-CoV-2, it has caused 10.2 million illnesses and 502 fatalities globally. With 2.6 million illnesses and 128 thousand deaths as of the end of July 2020, the United States of America is the most impacted country. COVID-19, abbreviation for coronavirus disease 2019, is caused by SARS-CoV-2. The daily status of COVID-19 is used as input in this investigation, using data published from John Hopkins University [28][42][43]. The input data analyzed is valid till May 27 and is passed into the neural networks shown below.

1. Recurrent Neural Network 2. Time Distributed LSTMs 3. Bidirectional LSTMs

The forecast is compared to the actual figures. The input dataset contains COVID-19 pandemic statistics for a period of 115 days and is provided as follows:

1. Worldwide Confirmed Cases, 2.Global Mortality, 3. Global Economic Recovery 4. Cases Confirmed in India, 5. India's Fatalities, 5. Indian Recoveries, 6. Confirmed Cases in the United States, 7. Deaths in the United States, 8. Recovery in the United States

Figure 8 shows the data point graph.

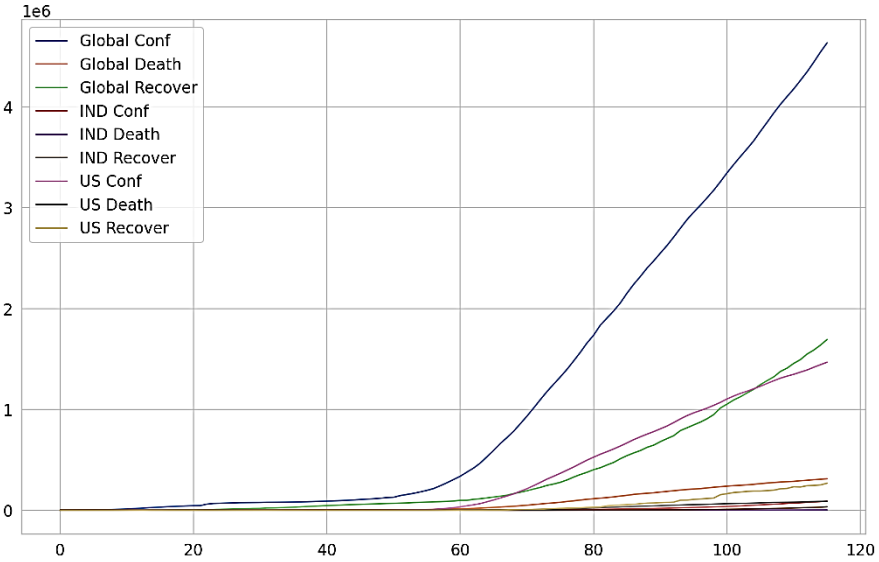

Fig. 8. COVID19 Data Plot (Actual Figures scaled by100000 on Y-axis)

The nine sequences stated above show correlation and can be used as input in both univariate and multivariate modes. They are fed into RNNs and LSTMs to anticipate future figures. To measure mistakes, these projections are compared to actual figures. For LSTM implementation, the Keras framework is used[26][44][45].

The Keras framework was created by Francois Chollet, a Developer/Researcher at Google AI, and used Theano as its default backend until version 1.1.0. Following that, Keras introduced backend support for TensorFlow, and with the release of Keras v1.1.0, TensorFlow became the default and most extensively used backend. Following the June 2019 launch of TensorFlow 2.0, Google recognized Keras as the official high-level TensorFlow API, facilitating model construction and training. Keras v2.3.0, released on September 17th, 2019, signified the synchronization of Keras with tf.keras[27][46][47].

3.1 LSTM Implementation using Keras

Implementation of any ANN model requires a set of process to be followed. Figure 9 depicts an overview of the seven processes in the artificial neural network model life-cycle in Keras, with five important steps, one data input stage, and one optimization step (hyper-parameter tweaking) included[28][48].

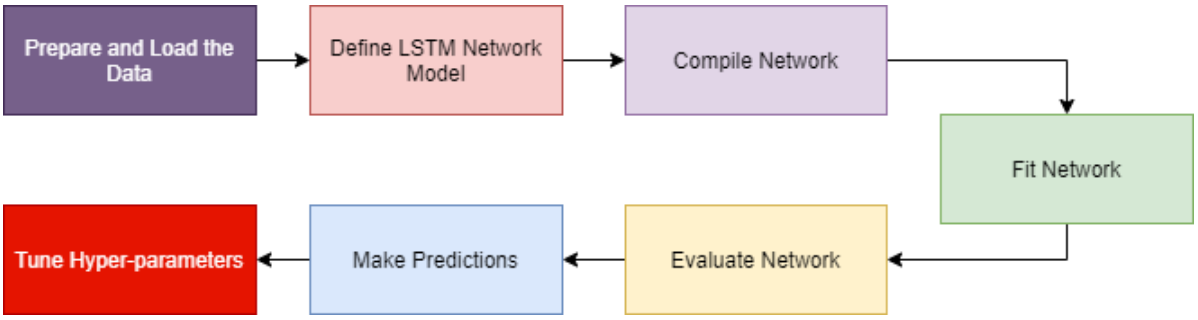

Fig. 9. Seven-stage Life Cycle for Neural Network Models

The life-cycle depicted above will be discussed in detail for two types of LSTMs namely Time Distributed LSTMs and Bidirectional LSTMs.

3.2 Time Distributed LSTM Model

The following discussion will centre on the sequential LSTM model. The sequential approach is suited for a simple stack of layers, with each layer containing exactly one input tensor and one output tensor. It represents a layer stack in a linear fashion. Although having well-defined and ostensibly user-friendly interfaces, such as those provided by the Keras framework, configuring and using LSTMs to random sequence prediction tasks can be difficult[29][49][50].

3.2.1 Preparing the data

The LSTM's recurrent layer, known as LSTM(), is made up of memory units, and it is followed by a fully connected layer known as Dense(), which serves as the output prediction layer. The input layer's shape is established by the first HL of the LSTM network, which also specifies the anticipated number of inputs. Three-dimensional (3D) input with the following samples, time steps, and attributes is required:

Samples: Represent the data's readings or rows.
Time steps: A sequence of earlier observations for a feature that captures the past time scale for that specific characteristic in a given sample.
Features: Correspond to data columns or properties. To comply with the LSTM specifications, the input must be moulded into a collection of [Sample, Time steps, Features].

As mentioned in section four, the data was initially provided as a 2D matrix having 115 rows and 9 columns. In the current prediction scenario, the challenge is given to perform multivariate and multistep sequence prediction. It involves taking in a large number of variables for input and forecasting a large number of variables as output. Each input sample has many time steps[30][51][52].

Scaling is deemed useful due to the high numeric values in the incoming data. As a result, the min-max scalar transformation is used to scale the data between 0 and 1. The data is reshaped in order to integrate time steps into

each sample. This entails taking into account the previous N measurements for each row and putting them together to generate a composite sample object. Out of the 115 samples, 88 are for training and 19 are for testing. With N=4 (number of time steps), the original input shape is [88, 9], and after 4 time steps of reshaping, it becomes [88, 4, 9].

3.2.1 Define the model

The TimeDistributed layers allow for the extraction of more complex time dependencies within samples. This wrapper allows you to apply a layer to each time step (temporal slice) of an input. As previously stated, the input should be at least 3D (sample, timestep, and features), with index one serving as the temporal dimension. The TimeDistributed wrapper layer must meet the following criteria:

1. The input must be three-dimensional - The final LSTM layer before the TimeDistributed wrapped Dense layer must be configured to return sequences (e.g., set return_sequences=True).
2. The output will be Three Dimensional- If the TimeDistributed wrapped Dense layer is the output layer predicting a sequence, the output or y array must be reshaped into a 3D vector with time steps matching the input sample time steps.

The "return_sequences" option must be set to true in order for the LSTM hidden layer to be configured to return sequences instead of single values.

```
# create LSTM
model = tf.keras.Sequential()
model.add(layers.LSTM(neurons, activation='relu',input_shape=(n_steps, n_features), return_sequences=True))
```

As a result, each LSTM unit will generate a series of n_steps outputs, one for each time step in the input data. On the output layer, the TimeDistributed layer is used to act as a fully interconnected Dense layer having an output corresponding to n_features.

```
model.add(layers.TimeDistributed(layers.Dense(n_features)))
```

The presence of n_features outputs in the output layer is critical; these represent the output data columns. Given that this is a multivariate prediction issue, the numerous outputs represent the aim to create one time step from the sequence for each time step in the input. This is accomplished by the TimeDistributed layer, which processes n_steps time steps of the input sequence at the same time.

One time step at a time, the TimeDistributed layer applies the same Dense layer (with the same weights) to the LSTM outputs. This technique allows the output layer to connect to each LSTM cell with only one connection. This technique therefore decreases network capacity, requiring an increase in the quantity of training epochs.

The full model definition is given below:

```
# define LSTM configuration
import tensorflow as tf
from tensorflow.keras import layers
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
# create LSTM
model = tf.keras.Sequential()
model.add (layers.LSTM(neurons, activation='relu',input_shape=(n_steps, n_features), return_sequences=True))
model.add(layers.TimeDistributed(layers.Dense(n_features)))
algorithm = tf.keras.optimizers.Adam(learning_rate=0.001,beta_1=0.9, beta_2=0.999)
model. compile(optimizer=algorithm, loss= tf.keras.losses.MeanSquaredError())
print(model. summary ())
```

3.2.3 Compile the model

Compiling is a necessary step after defining the model. It is required both before using an optimisation technique for training as well as after loading a set of pre-trained weights from a save file. The main reason for doing this is that compilation creates an effective representation of the network, that is necessary for generating predictions on your hardware.

The completely linked or Dense() output layer is crucial. This layer precisely matches with the one-to-one mapping of sample time steps. Each neurone in this layer corresponds to one weight and bias for each LSTM unit in the preceding layer. As a result, the problem is framed and learned in line with its definition, keeping a one-to-one mapping between input and output and segregating the internal process for each time step. Because the dense layer processes one time step at a time, it also contributes to network simplification by needing fewer weights. The network's compilation yields the following outcomes:

Model: "sequential_1"

Layer (type)	Output Shape	Param #
lstm_3 (LSTM)	(None, 4, 100)	44000
time_distributed (TimeDistri	(None, 4, 9)	909

Total params: 44,909, Trainable params: 44,909 and Non-trainable params: 0

There are 44,909 parameters in the LSTM layer. This computation is generated from the number of inputs (9, indicating the number of features) and the number of outputs (100, representing the 100 units in the hidden layer), which are computed as follows:

n1 = 4 * ((inputs + 1) * outputs + outputs^2)
n1 = 4 * ((9 + 1) *100 + 100^2)
n1 = 4 * 11000
n1 = 44000

Furthermore, the fully connected layer has only 9 parameters: the number of inputs (100, reflecting the preceding layer's 100 inputs), the number of outputs (1, corresponding to the layer's single neurone), and the bias.

n2 = inputs * outputs + outputs
n2 = 100 * 9 + 9
n2 = 909
total parameters = n = n1+ n2 = 44,909

3.2.5 Fit and Evaluate the model

Once the compilation is complete, the network can be fitted, which involves adjusting weights on the training dataset. This fitting step necessitates specifying the training data and ensuring that it corresponds to the input and output data shapes defined during the model's construction and compilation. The backpropagation technique is used to update the network throughout the training process. Following that, optimisation is performed using the selected optimisation algorithm and loss function from the compilation process. The backpropagation algorithm iteratively trains the network on the training dataset for a set number of cycles or epochs. The following commands are used to carry out this training:

# fit model
history = model.fit(X, Y,batch_size=n_batch, epochs=n_epoch, verbose=0)
The network can be evaluated on the training data, but as it has seen all the training data before, better insights are possible only on the testing data. The model performance is evaluated to estimate the loss across the test patterns. Other metrics such as classification accuracy are also the possible output of network evaluation process.

```
# evaluate model
loss = model.evaluate(Xt, Yt, verbose=2)
print ("MAE: %f" %loss)
```

Output:

```
1/1 - 0s - loss: 0.0097
MAE: 0.009734
```

The model loss plot is given below in Figure 10 as a result of the training process.

3.2.6 Make Predictions

When the model has reached the required level of performance, it can be used to predict outcomes on new data by executing the predict() method on the model with an array of new input patterns. A example input data visualisation is shown below, exhibiting a single normalised sample with four time steps and nine columns:

```
Shape :((1, 4, 9),
array([[[0.8298591 , 0.8645963 , 0.7587625 , 0.6216464 , 0.65795887, 0.49018234, 0.8563877 , 0.85249114,
0.7267267 ], [0.84978914, 0.8816958 , 0.7807975 , 0.6585363 , 0.6913967, 0.52264494, 0.87471825, 0.86959463,
0.74147093], [0.8683377 , 0.8958507 , 0.8124387 , 0.69287795, 0.73180073, 0.56396097, 0.8921734 , 0.887791
, 0.79192626], [0.88510466, 0.90675 , 0.8321391 , 0.7408989 , 0.7704632 , 0.6126987 , 0.9056015 , 0.8960272
, 0.8054707 ]]], dtype=float32))
# demonstrate prediction
yhat = model.predict(x_input, verbose=0)
yhat,yhat.shape
```

Output:

```
(array ([[[0.76646 , 0.9782676 , 0.77557725, 0.6167766 , 0.58303535, 0.38767594, 0.8497515 , 1.0831983 ,
0.8218566 ], [0.9142243 , 0.98353875, 0.8136648 , 0.64108723, 0.5122521 , 0.53985286, 0.8494971 , 1.0338019
, 0.8567422 ], [0.9716994 , 1.0278854 , 0.8781567 , 0.7336014 , 0.547858 , 0.6684656 , 0.87964684, 1.1155745 ,
0.9724355 ], [1.0361695 , 1.0794674 , 0.942116 , 0.806292 , 0.5749702 , 0.78490067, 0.91305393, 1.194532 ,
1.080696 ]]], dtype=float32), (1, 4, 9))
```

The output is the next set of COVID19 cases for the given input, it gives the three-dimensional output. Further the predictions are calculated for set of inputs and plotted on the actual graph, it can be seen in Figure 11, further the predictions were also made by a simple LSTM network with two HLs, and its output shown in Figure 12, the Time distributed LSTMs have given better prediction.

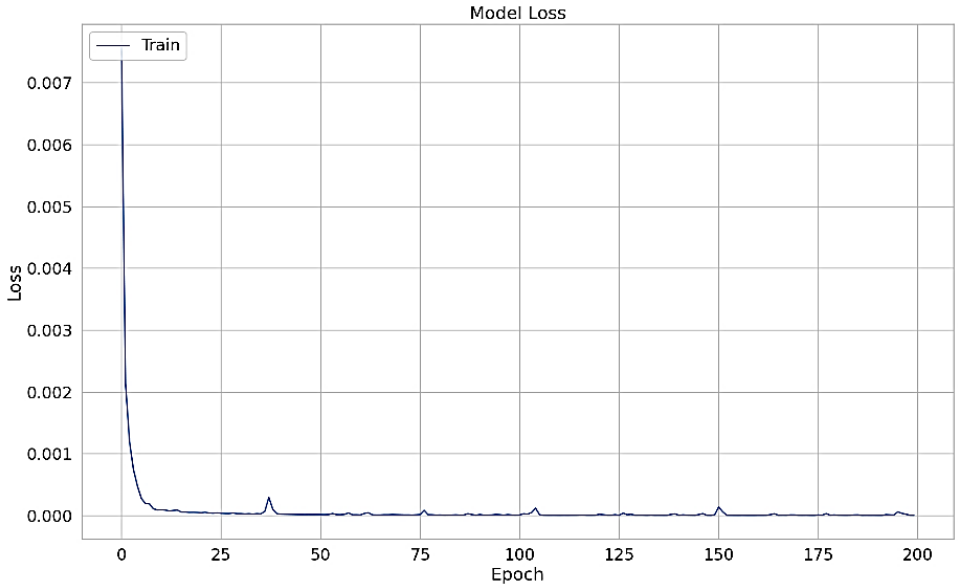

Fig. 10. Output of the Network training

3.3 Tune Hyper-Parameters

Machine learning models are parameterized and for a given problem their performance can be tuned. These models do have many parameters and to search for the best combination of parameters is a complex problem with a number of valid solutions.

Parameter for a machine learning model- A ML model parameter is an internal configuration parameters that can be estimated from the dataset. Further,

- When making predictions, they play a key role
- They define the performance of the model for a given problem
- Many times, manual setting is not allowed for them
- These parameters are part of the learned model.

Hyperparameter for a machine learning model- An external configuration variable that cannot be predicted from data is referred to as a model hyperparameter. These parameters, which are manually given by the practitioner, are used by processes to assist in determining the model parameters. They are refined using heuristics or empirical methods to optimise the output of a specific predictive modelling challenge.

For the Long Short-Term Memory (LSTM) discussed in this chapter, there are lots of hyperparameters, (learning rate, hidden units, batch size etc.) for which one has to get the optimum combination. Depending on the size of a machine learning model, the hyperparameter tuning generally takes a considerable amount of time.
The performance of these sequence classifiers is critically based on their hyperparameters, and it is crucial to implement an efficient method to estimate the optimal values. The hyper parameter tuning in machine learning is done using a separate set of validation data set. The hyperparameter is tuned by searching in a range for the given parameter and those set of parameters are selected which result in the best accuracy on the validation set [30]. The Hyperparameters applicable for the LSTM Models are listed below.

1. Number of Neurons in the layer 2. Batch Size of the data to be fed to the model
1. Number of training epochs 4. Dropout - The regularization methods like dropout which slows down the learning Layers 5. Regularization- To avoid the overfitting of the
6. Optimization Algorithms 7. Loss Functions 8. Features and Timesteps.

Various combinations are possible and this a whole separate domain of research, here the output of the one of the best performing models are presented in Figure 11 and Figure 12 for the comparison purpose and further it can be explored.

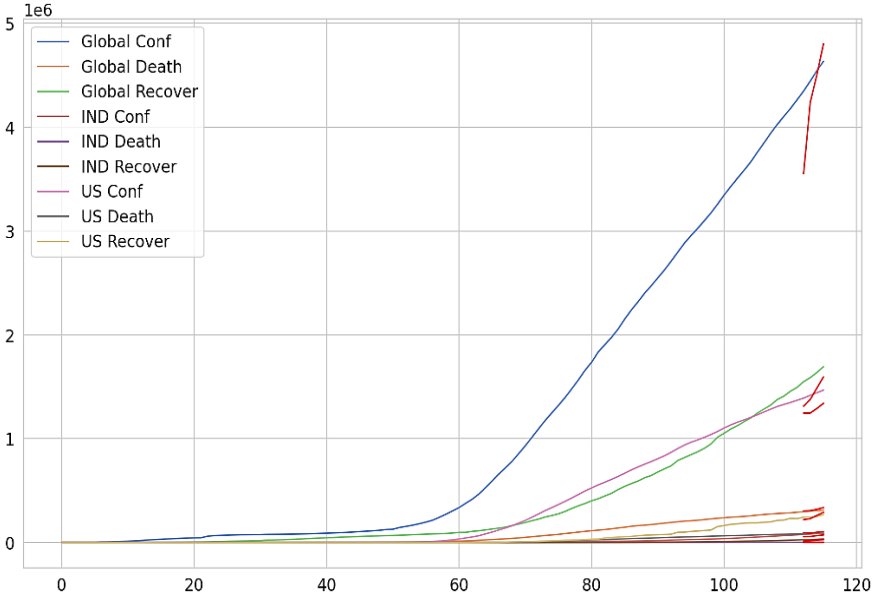

Fig. 11. The forecasting of COVID-19 cases using Time Distributed LSTMs is depicted in red markings, overlaid on the actual values.

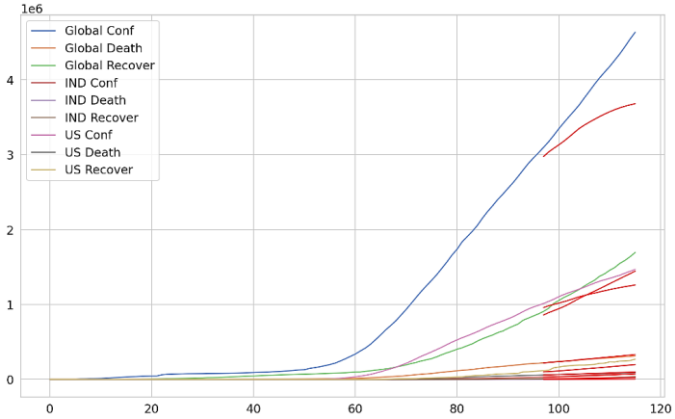

Fig. 12. The prediction of COVID-19 cases using Regular LSTMs is indicated by red markings superimposed on the actual values.

3.4 Bidirectional LSTM Model

LSTMs maintain information from the inputs that they have navigated through hidden states. Bidirectional LSTMs (BiLSTMs) are LSTM extensions that improve model performance in sequence classification. Unlike Unidirectional LSTMs, which only maintain knowledge from the past because they only face inputs from previous time steps, BiLSTMs process inputs in both directions—from the past to the future and from the future to the past. In BiLSTMs, the backward-running LSTM maintains information from the future, allowing for the preservation of both past and future information at any given time. This distinguishing trait puts BiLSTMs as effective context-understanding tools, resulting in remarkable results. When the whole input sequence is accessible, BiLSTMs train two LSTMs, one on the original sequence and the other on a reversed copy. This tactic gives the network more context, which enables quicker and more thorough issue solving [31].

3.4.1 Bidirectional LSTMs in Keras

Keras has a feature that allows bidirectional LSTMs called the Bidirectional layer wrapper. As an input, this wrapper expects the first LSTM layer (recurrent layer). It enables the definition of the merging mode, which specifies how the forwards and backwards outputs should be blended before being passed to the next layer. Merge modes include the following:

Sum: The outcomes are totaled.
Mul: The results are combined and multiplied.
Concat: The outputs are concatenated, yielding twice as many outputs for the next layer (the default setting).
Avg: The outputs are averaged.

Following Code defines the BiLSTMs in Keras

```
# define LSTM configuration
import tensorflow as tf
from tensorflow.keras import layers
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
model = tf.keras.Sequential()
model.add(layers.Bidirectional(layers.LSTM(neurons, activation='relu', return_sequences=True), input_shape=(n_steps, n_f
eatures)))
model.add(layers.TimeDistributed(layers.Dense(n_features)))
algorithm = tf.keras.optimizers.Adam(learning_rate=0.001,beta_1=0.9, beta_2=0.999)
model.compile(optimizer=algorithm, loss= tf.keras.losses.MeanSquaredError())
model.summary()
```

Output after Compilation is as follows:

Model: "sequential_3"

Layer (type)	Output Shape	Param #
bidirectional_3 (Bidirection	(None, 4, 200)	88000
time_distributed_3 (TimeDist	(None, 4, 9)	1809

Total params: 89,809 , Trainable params: 89,809 and Non-trainable params: 0

It is clear that the BiLSTM layer has 89,809 parameters. This computation depends on the number of inputs (9, indicating the total features), the number of layers (2, accounting for forwards and backwards), and the total outputs (100, representing the 100 units in the HL), as shown below:

n1 = 4 * 2 * ((inputs + 1) * outputs + outputs^2)
n1 = 4 * 2 * ((9 + 1) *100 + 100^2)
n1 = 4 * 2 * 11000
n1 = 88,000

Furthermore, the fully connected layer contains only 9 parameters: the total inputs (100, reflecting the 100 inputs from the preceding layer), the total outputs (1, corresponding to the layer's single neurone), and the bias.

n2 = inputs * outputs + inputs * outputs + outputs
n2 = 9 * 100 + 9 * 100 + 9
n2 = 1809

total parameters = n = n1+ n2 = 89,809

The prediction of the BiLSTMS is shown in Figure 12. Further for the individual LSTMs i.e. the forward, the backward and the combined are evaluated separately and their training performance is generated. The loss plot is shown in Figure 13. The log loss for LSTM forwards (blue) and LSTM backwards (orange) display similar patterns during the 250 training epochs, as seen in the results. In comparison to the other two configurations, the Bidirectional LSTM log loss (green) varies by attaining a lower value sooner and sustaining an overall lower value.

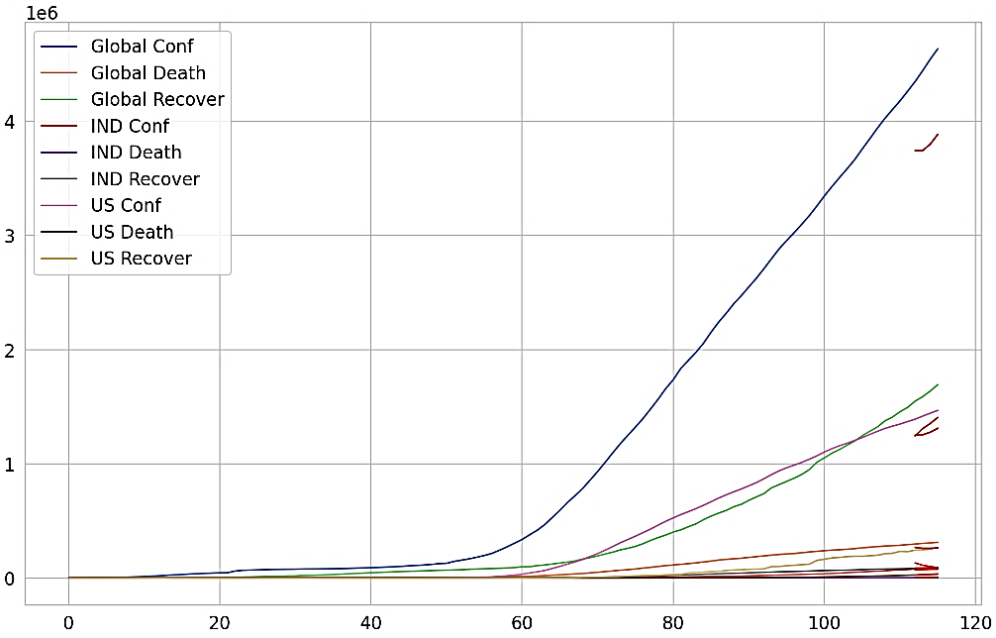

Fig. 13. The prediction of COVID-19 cases using Bidirectional LSTMs is illustrated by red markings overlaid on the actual values.

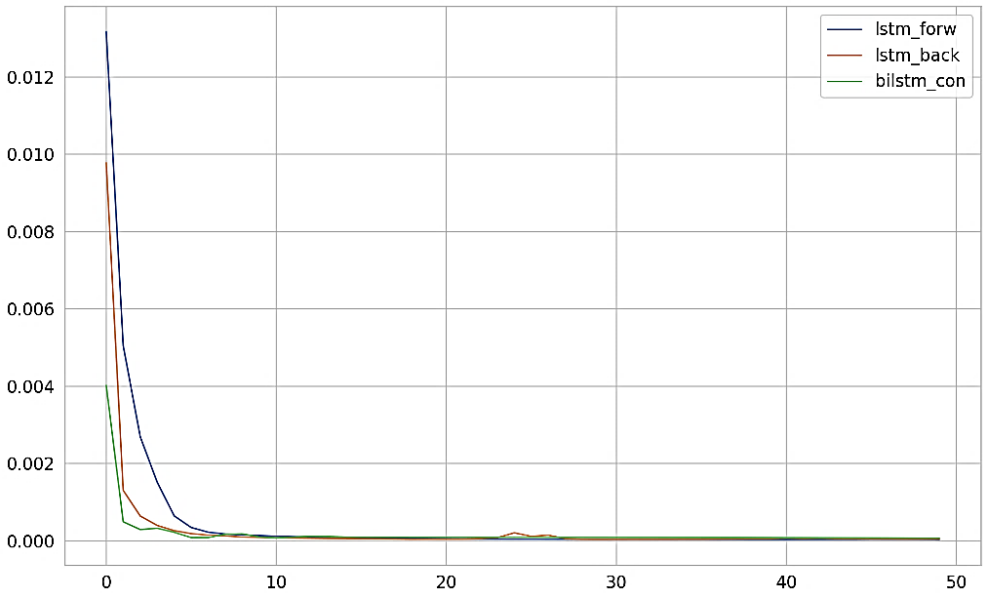

Fig. 14. Performance comparison forward, backward and Bidirectional LSTM layer.

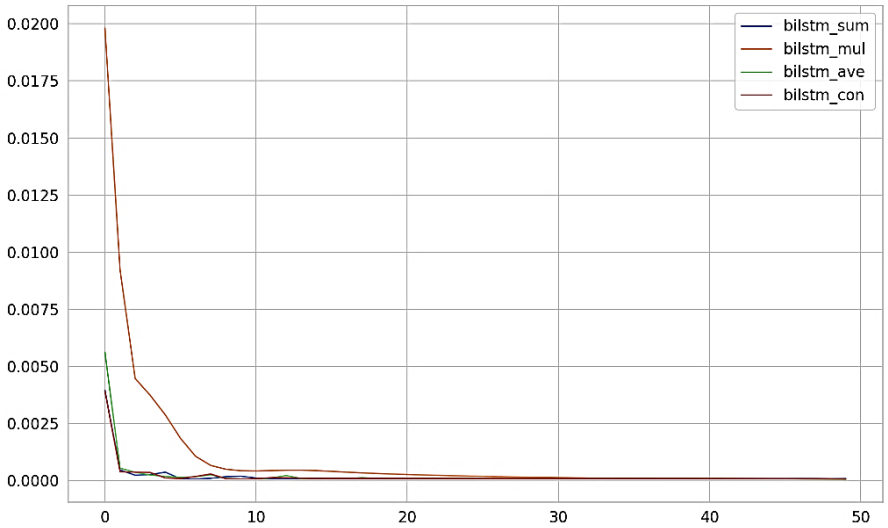

Fig. 15. Performance comparison different Bidirectional LSTMs merge modes.

As previously stated, four distinct merging mechanisms are available for integrating the results of Bidirectional LSTM layers: concatenation (the default), multiplication, average, and sum. To evaluate performance, an example from the previous section was updated, and the result is shown in Figure 15. The comparison shows that the sum (in blue) and concatenation (in red) merging modes may produce superior results, or at the very least a reduced log loss. The following part will compare the outcomes received by the aforementioned LSTMs and RNNs.

4. RESULT DISCUSSION AND CONCLUSION

The basic or conventional RNN, often known as vanilla RNN, was created from the ground up. As input, it was given a single series of worldwide confirmed cases. Because the RNN is a vanilla type, its adaptability to multivariate data is limited, with only minor adjustments possible. Figure 16 depicts the input data, where the RNN is trained on the sequence data of global confirmed cases, predicting the next number based on the given input. Figure 17 shows the RNN predictions, which exhibit significant departures from both the actual values and the trendline for the sequence.

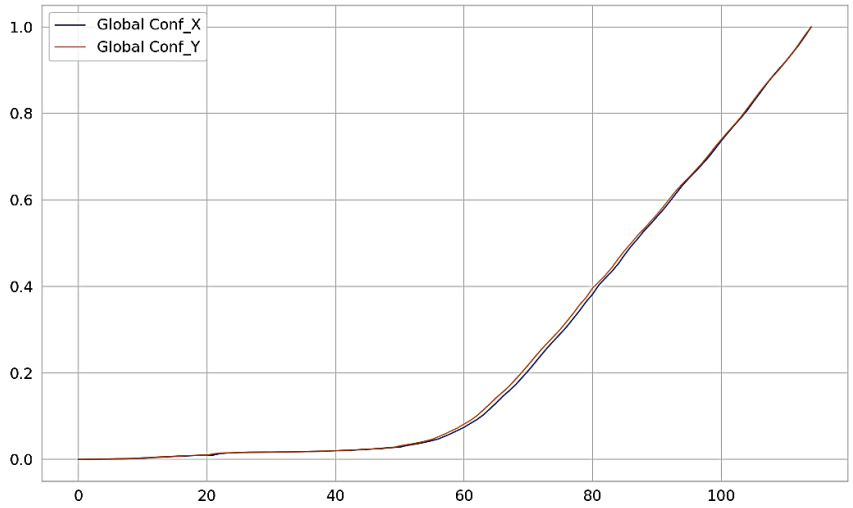

Fig. 16. Univariate sequence data input for global confirmed COVID-19 cases.

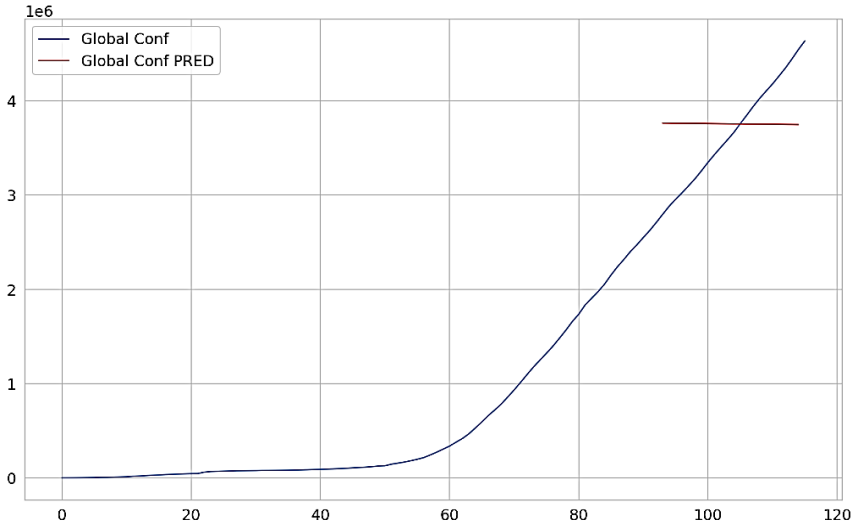

Figure 17 The RNN's predictions for univariate sequence data of global confirmed COVID-19 cases are depicted in red markings overlaid on the actual values.

Finally, the performance of all the models studied in this chapter is given in Figure 18, where it can be observed that the LSTMs outperform the vanilla RNN.

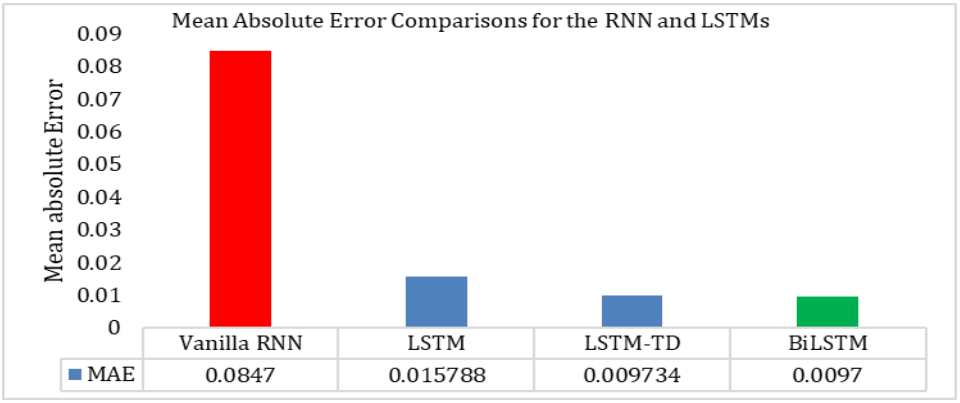

Fig. 18. Comparing the performance of RNN and LSTMs (Red bar signifies the lowest performance, while the Green bar represents the highest performance).

In the various models of LSTMs, the performance is almost in close range, however the TimeDistributed LSTMs and Bidirectional LSTMs perform better than the conventional LSTMs for the given problem of the COVID19 sequence prediction. The implementation of all the RNN and LSTM models mentioned above is available for download at DOI:10.5281/zenodo.3928612 [32]. This research looks into deep learning using RNNs. LSTM Networks are offered as a remedy to the vanishing and ballooning gradient issues that standard RNNs face. The Keras framework is used to investigate the development of time-distributed and bidirectional LSTMs. LSTMs and its variations are used to forecast COVID-19 instances. The results show that LSTMs outperform RNNs in terms of performance, with Bidirectional LSTMs performing even better.

5. REFERENCE

[1] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," Bulletin of Mathematical Biophysics, vol. 5, pp. 115–133, 1943. DOI: 10.1007/BF02478259
[2] D. O. Hebb, "The organization of behavior; a neuropsychological theory," Wiley, 1949.
[3] F. Rosenblatt, "The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain," Psychological Review, vol. 65, no. 6, pp. 386–408, 1958. DOI: 10.1037/h0042519
[4] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation," in Readings in Cognitive Science: A Perspective from Psychology and Artificial Intelligence, vol. 1, pp. 333–336, 1986.
[5] G. E. Hinton and S. Osindero, "A Fast Learning Algorithm for Deep Belief Nets," Yee-Whye Teh.
[6] A. K. Jain, J. Mao, and K. M. Mohiuddin, "Artificial neural networks: A tutorial," Computer, vol. 29, pp. 31–44, 1996.
[7] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in Advances in Neural Information Processing Systems, pp. 153–160, 2007.
[8] B. Nemade and D. Shah, “An IoT based efficient Air pollution prediction system using DLMNN classifier,” in Physics and Chemistry of the Earth, Parts A/B/C, vol. 128, p. 103242, 2022. [Online]. Available: https://doi.org/10.1016/j.pce.2022.103242
[9] B. Nemade and D. Shah, “An efficient IoT based prediction system for classification of water using novel adaptive incremental learning framework,” in Journal of King Saud University-Computer and Information Sciences, vol. 34, no. 8, pp. 5121-5131, 2022. [Online]. Available: https://doi.org/10.1016/j.jksuci.2022.01.009
[10] S. Badillo, B. Banfai, F. Birzele, I. I. Davydov, L. Hutchinson, T. Kam-Thong, and J. D. Zhang, "An Introduction to Machine Learning," Clinical Pharmacology and Therapeutics, vol. 107, no. 4, pp. 871–885, 2020. DOI: 10.1002/cpt.1796.
[11] A. Chahal and P. Gulia, "Machine learning and deep learning," International Journal of Innovative Technology and Exploring Engineering, vol. 8, no. 12, pp. 4910–4914, 2019. DOI: 10.35940/ijitee.L3550.1081219.
[12] Y. Lecun, Y. Bengio, and G. Hinton, "Deep learning," Nature, Nature Publishing Group, 2015. DOI: 10.1038/nature14539.
[13] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," Science, American Association for the Advancement of Science, 2015. DOI: 10.1126/science.aaa8415.
[14] M. D. Landry, L. Geddes, A. Park Moseman, J. P. Lefler, S. R. Raman, and J. van Wijchen, "Early reflection on the global impact of COVID19, and implications for physiotherapy," Physiotherapy (United Kingdom), Elsevier Ltd, 2020. DOI: 10.1016/j.physio.2020.03.003.
[15] J. Martens and I. Sutskever, "Training Deep and Recurrent Networks," Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 7700 LECTU, pp. 1–58, 2012. DOI: 10.1007/978-3-642-35289-8-27.
[16] S. L. Ho, M. Xie, and T. N. Goh, "A comparative study of neural network and Box-Jenkins ARIMA modeling in time series prediction," Computers and Industrial Engineering, vol. 42, pp. 371–375, 2002. DOI: 10.1016/S0360-8352(02)00036-0.
[17] M. Sundermeyer, R. Schlüter, and H. Ney, "LSTM neural networks for language modeling," in 13th Annual Conference of the International Speech Communication Association 2012, INTERSPEECH 2012, vol. 1, pp. 194–197, 2012.
[18] Q. Song, Y. Wu, and Y. C. Soh, "Robust adaptive gradient-descent training algorithm for recurrent neural networks in discrete time domain," IEEE Transactions on Neural Networks, vol. 19, no. 11, pp. 1841–1853, 2008. DOI: 10.1109/TNN.2008.2001923.

[19] J. C. Principe, J. M. Kuo, and B. De Vries, "Backpropagation through time with fixed memory size requirements," in Neural Networks for Signal Processing III - Proceedings of the 1993 IEEE Workshop, NNSP 1993, pp. 207–215, 1993. DOI: 10.1109/NNSP.1993.471868.
[20] S. Squartini, A. Hussain, and F. Piazza, "Attempting to Reduce the Vanishing Gradient Effect through a novel Recurrent Multiscale Architecture," in Proceedings of the International Joint Conference on Neural Networks, vol. 4, pp. 2819–2824, 2003. DOI: 10.1109/ijcnn.2003.1224018.
[21] A. Zeyer, P. Doetsch, P. Voigtlaender, R. Schluter, and H. Ney, "A comprehensive study of deep bidirectional LSTM RNNS for acoustic modeling in speech recognition," in ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings, pp. 2462–2466, 2017. DOI: 10.1109/ICASSP.2017.7952599.
[22] S. Hochreiter and Jürgen Schmidhuber, "LSTM can solve hard long time lag problems," Proceedings of the 9th International Conference on Neural Information Processing Systems, pp. 473–479, 1996. DOI: 10.5555/2998981.2999048.
[23] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling." http://arxiv.org/abs/1412.3555.
[24] M. Collier and J. Beel, "Implementing Neural Turing Machines," in Artificial Neural Networks and Machine Learning – ICANN 2018, Lecture Notes in Computer Science, vol. 11141, Springer, Cham, 2018.
[25] R. DiPietro and G. D. Hager, "Deep learning: RNNs and LSTM," in Handbook of Medical Image Computing and Computer Assisted Intervention, pp. 503–519, Elsevier, 2019. DOI: 10.1016/B978-0-12-816176-0.00026-0.
[26] Adrian Rosebrock, "Keras vs. tf.keras: What's the difference in TensorFlow 2.0?" - PyImageSearch. https://www.pyimagesearch.com/2019/10/21/keras-vs-tf-keras-whats-the-difference-in-tensorflow-2-0/.
[27] H. Li, Z. Liu, and J. Ge, "Scientific research progress of COVID-19/ SARS-CoV-2 in the first five months," Journal of Cellular and Molecular Medicine, Blackwell Publishing Inc, 2020. DOI: 10.1111/jcmm.15364.
[28] E. Dong, H. Du, and L. Gardner, "An interactive web-based dashboard to track COVID-19 in real time," The Lancet Infectious Diseases, 2020. DOI: 10.1016/S1473-3099(20)30120-1.
[29] L. Capelo, "Beginning Application Development with TensorFlow and Keras: Learn to Design, Develop, Train, and Deploy TensorFlow and Keras Models as Real-World Applications (1st ed.)," Packt Publishing, 2018. DOI: 10.5555/3265575.
[30] B. Nakisa, M. N. Rastgoo, A. Rakotonirainy, F. Maire, and V. Chandran, "Long short term memory hyperparameter optimization for a neural network based emotion recognition framework," IEEE Access, vol. 6, pp. 49325–49338, 2018. DOI: 10.1109/ACCESS.2018.2868361.
[31] M. Schuster and K. K. Paliwal, "Bidirectional Recurrent Neural Networks," IEEE TRANSACTIONS ON SIGNAL PROCESSING, vol. 45, no. 11.
[32] Vinayak Bharadi, "Using Long Short-Term Memory (LSTM) Networks for COVID19 Impact Prediction," Zenodo. https://doi.org/10.5281/zenodo.3928612.
[33] M. Shirodkar, V. Sinha, U. Jain, and B. Nemade, "Automated attendance management system using face recognition," in International Journal of Computer Applications, vol. 975, 2015.
[34] K. Patra, B. Nemade, D. P. Mishra, and P. P. Satapathy, "Cued-click point graphical password using circular tolerance to increase password space and persuasive features," in Procedia Computer Science, vol. 79, pp. 561-568, 2016. [Online]. Available: https://doi.org/10.1016/j.procs.2016.03.071
[35] V. A. Bharadi, B. Pandya, and B. Nemade, "Multimodal biometric recognition using iris & fingerprint: By texture feature extraction using hybrid wavelets," in 2014 5th International Conference - Confluence The Next Generation Information Technology Summit (Confluence), Noida, India, 2014, pp. 697-702. doi: 10.1109/CONFLUENCE.2014.6949309.
[36] V. A. Bharadi, V. I. Singh, and B. Nemade, "Hybrid Wavelets based Feature Vector Generation from Multidimensional Data set for On-line Handwritten Signature Recognition," in 2014 5th International Conference - Confluence The Next Generation Information Technology Summit (Confluence), Noida, India, 2014, pp. 561-568. doi: 10.1109/CONFLUENCE.2014.6949038.
[37] R. Mishra, B. Nemade, K. Shah, and P. Jangid, "Improved Inductive Learning Approach-5 (IILA-5) in Distributed System," Int. J. Intell. Syst. Appl. Eng., vol. 11, no. 10s, pp. 942-953, 2023.
[38] B. C. Surve, B. Nemade, and V. Kaul, "Nano-electronic devices with in machine learning capabilities," ICTACT J. Microelectron., vol. 9, no. 3, pp. 1601-1606, 2023. doi: 10.21917/ijme.2023.0277
[39] B. Nemade, V. Bharadi, S. S. Alegavi, and B. Marakarkandy, "A Comprehensive Review: SMOTE-Based Oversampling Methods for Imbalanced Classification Techniques, Evaluation, and Result Comparisons," Int. J. Intell. Syst. Appl. Eng., vol. 11, no. 9s, pp. 790-803, 2023.

[40] B. P. Nemade, K. Shah, B. Marakarkandy, K. Shah, B. C. Surve, and R. K. Nagra, "An Efficient IoT-Based Automated Food Waste Management System with Food Spoilage Detection," Int. J. Intell. Syst. Appl. Eng., vol. 12, no. 5s, pp. 434-449, 2024.
[41] S. S. Alegavi, B. Nemade, V. Bharadi, S. Gupta, V. Singh, and A. Belge, "Revolutionizing Healthcare through Health Monitoring Applications with Wearable Biomedical Devices," Int. J. Recent Innov. Trends Comput. Commun., vol. 11, no. 9s, pp. 752–766, 2023. [Online]. Available: https://doi.org/10.17762/ijritcc.v11i9s.7890
[42] B. Nemade, N. Phadnis, A. Desai, and K. K. Mungekar, "Enhancing connectivity and intelligence through embedded Internet of Things devices," ICTACT Journal on Microelectronics, vol. 9, no. 4, pp. 1670-1674, Jan. 2024, doi: 10.21917/ijme.2024.0289.
[43] B. C. Surve, B. Nemade, and V. Kaul, "Nano-electronic devices with machine learning capabilities," ICTACT Journal on Microelectronics, vol. 9, no. 3, pp. 1601-1606, Oct. 2023, doi: 10.21917/ijme.2023.0277.
[44] G. Khandelwal, B. Nemade, N. Badhe, D. Mali, K. Gaikwad, and N. Ansari, "Designing and Developing novel methods for Enhancing the Accuracy of Water Quality Prediction for Aquaponic Farming," Advances in Nonlinear Variational Inequalities, vol. 27, no. 3, pp. 302-316, Aug. 2024, ISSN: 1092-910X.
[45] B. Nemade, S. S. Alegavi, N. B. Badhe, and A. Desai, "Enhancing information security in multimedia streams through logic learning machine assisted moth-flame optimization," ICTACT Journal of Communication Technology, vol. 14, no. 3, 2023.
[46] S. S. Alegavi, B. Nemade, V. Bharadi, S. Gupta, V. Singh, and A. Belge, "Revolutionizing Healthcare through Health Monitoring Applications with Wearable Biomedical Devices," International Journal of Recent Innovations and Trends in Computing and Communication, vol. 11, no. 9s, pp. 752–766, 2023. [Online]. Available: https://doi.org/10.17762/ijritcc.v11i9s.7890.
[47] V. Kulkarni, B. Nemade, S. Patel, K. Patel, and S. Velpula, "A short report on ADHD detection using convolutional neural networks," Frontiers in Psychiatry, vol. 15, p. 1426155, Sept. 2024, doi: 10.3389/fpsyt.2024.1426155.
[48] B. Nemade and D. Shah, "An IoT-Based Efficient Water Quality Prediction System for Aquaponics Farming," in Computational Intelligence: Select Proceedings of InCITe 2022, Singapore: Springer Nature Singapore, 2023, pp. 311-323. [Online]. Available: https://doi.org/10.1007/978-981-19-7346-8_27.
[49] B. Nemade and D. Shah, "IoT-based Water Parameter Testing in Linear Topology," in 2020 10th International Conference on Cloud Computing, Data Science and Engineering (Confluence), Noida, India, 2020, pp. 546-551, doi: 10.1109/Confluence47617.2020.9058224.
[50] B. Nemade and D. Shah, "An IoT based efficient Air pollution prediction system using DLMNN classifier," in Physics and Chemistry of the Earth, Parts A/B/C, vol. 128, p. 103242, 2022. [Online]. Available: https://doi.org/10.1016/j.pce.2022.103242
[51] B. Nemade and D. Shah, "An efficient IoT based prediction system for classification of water using novel adaptive incremental learning framework," in Journal of King Saud University-Computer and Information Sciences, vol. 34, no. 8, pp. 5121-5131, 2022. [Online]. Available: https://doi.org/10.1016/j.jksuci.2022.01.009
[52] V. Kaul, B. Nemade, and V. Bharadi, "Next Generation Encryption using Security Enhancement Algorithms for End to End Data Transmission in 3G/4G Networks," in Procedia Computer Science, vol. 79, pp. 1051-1059, 2016. [Online]. Available: https://doi.org/10.1016/j.procs.2016.03.133