Deep Learning Approaches for Anomaly Detection in Time Series

Chinmay D Hegde, Divya V Naik

Cologne University of Applied Science

07-02-2019

Abstract. Anomaly detection in time series refers to the finding of outliers in the time series pattern. In this case study, the main aim is to acquire the necessary knowledge to learn deep learning approaches for anomaly detection in time series. In this process, we studied different deep learning networks like Recurrent Neural Network, Long Short Term Memory, Convolutional Neural Network, Gated Recurrent Unit, and Autoencoder. All these were applied by TensorFlow and Keras platform. Mainly LSTM was used in several variants of Sine wave prediction. The next approach was to study the scientific paper -"P. Malhotra, L. Vig, G. Shro, and P. Agarwal, Long Short Term Memory Networks for Anomaly Detection in Time Series,". The main idea of the paper was to detect anomaly in the time series using stacked LSTM networks. Few critical points regarding the paper were discussed. They claimed that the LSTM was successfully implemented to detect anomalies in various time series data. We implemented the same steps for power data set. The algorithm was able to detect anomaly in certain expected regions. There were also false predictions. The tuning of the model can be considered as the future scope of the case study.
.

Keywords: RNN, LSTM, GRU, CNN, autoencoder, anomaly detection, time series, deep learning

Acknowledgment

First and foremost, we would like to thank our guide Prof Wolfgang Konen and Markus Thill for their valuable ideas, feedback and guidance throughout the case studies. With their guidance, we have done our study in the right approach. Thanks for clearing our doubts and supporting us throughout the case studies.

Table of Contents

Abbreviations

1. DNN - Deep Neural Network
2. RNN - Recurrent Neural Network
3. LSTM - Long Short Term Memory
4. GRU - Gated Recurrent Unit
5. ReLU - Rectified Linear Unit
6. MSE - Mean Squared Error
7. PD - Probability Density
8. MLE - Maximum Likelihood Estimator

1 Introduction

Rina Dechter in 1986 introduced the term Deep Learning and Igor Aizenberg in 2000 introduced artificial neural network[1]. Deep learning helps in many fields like object identification, speech to text conversion, recommendations of new products, pattern recognition and have made remarkable improvements in the state-of-the-art[2]. Artificial neural networks are motivated by the biological neuron structure and convolutional feedforward networks by the architecture of visual hierarchy[3].

In this case study, we have explored the Deep Neural Network (DNN) and gained the knowledge about different deep learning algorithms. We initially started with basic neural network that is Recurrent Neural Network (RNN), its architecture and its importance. Next we continued our studies on Long Short Term Memory (LSTM). Then we studied Gated Recurrent Unit (GRU). Later we have done research on CNN, autoencoder and its architecture. We studied the LSTM tutorials of implementation of MNIST dataset and done some analysis. Then we implemented predicting Sine wave on Keras. Later we studied the scientific paper on Long Short Term Memory networks for anomaly detection in time series, discussed their paper and finally, we implemented the paper.

The rest of the report is organized as follows: In Section 2 we give an overview of Deep Neural Networks which includes RNN, LSTM, GRU, CNN and autoencoder, and its architecture and remarks. In Section 3 we have covered a brief overview of TensorFlow and Keras. Next, we implemented MNIST data in LSTM network and autoencoder, discussion of errors and solutions in Section 4. Then we introduce the implementation of a sine wave in Keras and TensorFlow in Section 5. This includes the implementation of sine wave prediction with basic LSTM along with further modifications and developments. Then we present our idea on paper Long Short Term Memory for anomaly detection in Time Series and its implementations. Finally, we conclude the report with discussions and Appendix work in Section 7 and 8 respectively.

2 Deep Neural Networks

Deep neural network appears as one of the most approved machine learning technique. A simple neural network takes input, performs different actions by passing information through neurons and gives the output that can be used in classification and regression[4]. A Deep neural network consists of more than one hidden layers. Every successive layer uses the output of the previous layer. It is a cascade of multiple layers of nonlinear processing units that is used for feature extraction and transformation. Here a diagram of what one node looks like is shown in Fig 1.

Deep Neural Network consists of input layers, output layer and more than two hidden layers. This is showed in Fig 2

1. The input units represent the information fed into the network.

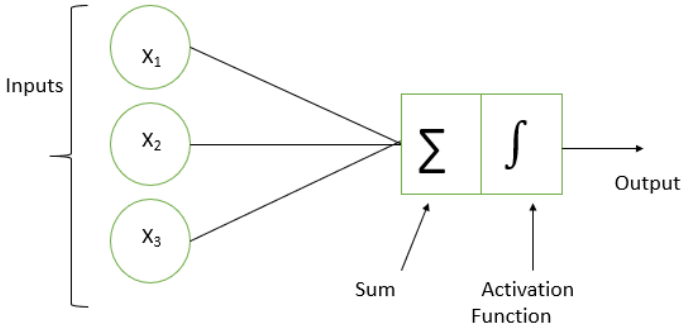


Fig. 1. One node of deep neural network with inputs, output, summing and activation function

2. The activity of the hidden layer is defined by input units and weights given between input and hidden units
3. The output units depend on input and hidden units. Fig 2. shows multi-layer architecture.

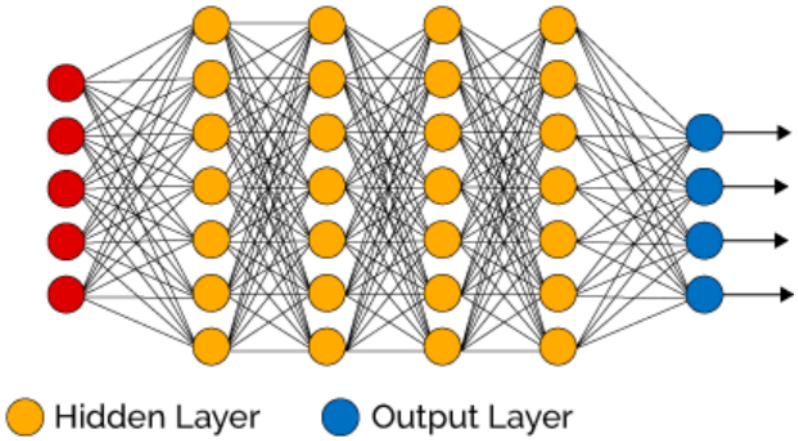


Fig. 2. Structure of Deep Neural Network with 4 hidden layers. Source: Deep Learning made easy with Deep Cognition

Activation Function: It is a transfer function used to determine the output of the neural network which maps the resulting values between 0 and 1 or -1 and 1. The Activation Functions can be divided into Linear Activation function and Non- linear Activation functions. For linear activation function, the final

output is directly proportional to the total weighted output. Threshold units, Rectified Linear Unit(ReLU) and Sigmoid units are the activation functions which are included in Non-linear activation function. For threshold units, the output always depend on the input, whether it is great or less to the particular threshold unit. In ReLU, the output will be either positive value or zero. For sigmoid units, the output changes continuously non-linearly with inputs[5].

2.1 RNN

Recurrent Neural Networks were introduced by David Rumelhart in 1986[6]. RNNs are class of artificial neural network as they do the same task for every element of a sequence and the output depends only on current input. Information is sent sequentially through the network. RNNs use its internal state for the processing of input. These are designed to work with sequence prediction problems. It considers the previous word during prediction, it acts like a memory unit which stores for a short period. All these makes its applications in speech recognition, machine translation or time series prediction[12].

These networks have shown great results in NLP tasks. In RNNs there are many variants which include fully recurrent, second order RNNs, bidirectional RNN, long short term memory, gated recurrent unit etc. The architecture of fully recurrent is explained in the next section. Between all these, LSTMs are prevalent and have many benefits in them due to it's long-term sequences than RNNs. RNNS have its applications in many fields:

1. Language Modeling - Predicting the likelihood of the next word.
2. Speech Recognition - Here predicting phonetic segments based on input sound waves.
3. Machine Translation - This includes translation of word or sentence from one language to another language.
4. Generating Image Descriptions - This application is a combination of RNN and CNN, where CNN helps for segmentation and RNN used to create a description for the segmented image.

2.2.1 RNN Architecture

A usual RNN has a short-term memory. The image below demonstrates an unrolled RNN. In the image, we can observe the RNN, which is folded before equals sign and unfolded after the equals sign. We can clearly observe that there is no feedback loop after equal signs which shows the information passed in one direction only. Here the parameters are passed from initial steps to final time steps[12].

As feed-forward neural networks consider only present input, not the past values, these are not so good at predicting the next values. This network sends the information recurrently.

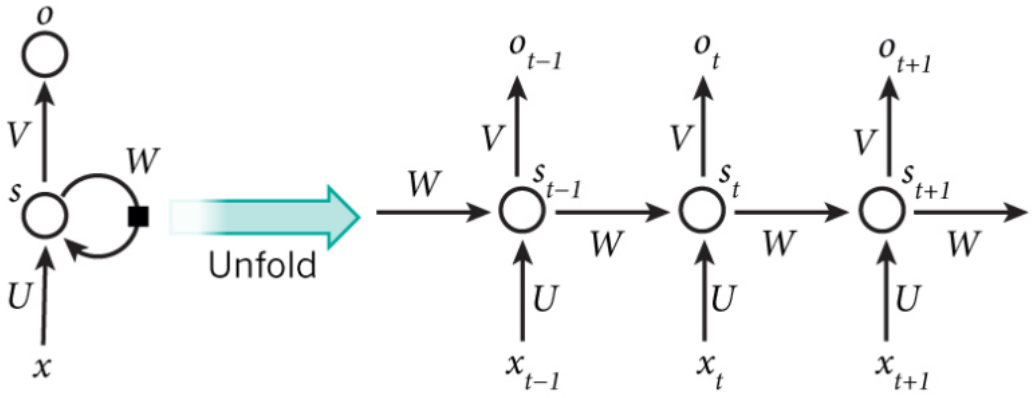


Fig. 3. A recurrent neural network and the unrolled architecture. Source: WILD ML Artificial Intelligence, Deep Learning, and NLP

The above figure depicts the RNN which is unrolled into a full network. The equations of the computations occur in RNN are as follows:

1. x_t is the input at timestep t.
2. s_t is the hidden state at timestep t. This is the memory cell depends on the previous hidden layer and the input.
3. $$s_t = f(Ux_t + W_t - 1)$$
   here f function is generally a nonlinear functions which are tanh or ReLU.
4. o_t is the output at time step t

2.1.2 Remarks on RNN

1. The problems found in RNN is vanishing gradient problem. Here when the gradient will be small and the weight updation becomes unnoticeable. This is solved with the concept of LSTM[25].
2. The hidden state s_t captures information about what happened in all previous time steps. It cannot capture too many time-steps.

2.2 LSTM

Hochreiter and Schmididhuber in 1997 proposed Long Short-Term Memory (LSTM) [7]. The main advantage of LSTM is to solve problem occured in RNN, that is vanishing gradient. LSTM has memory cell which helps to track long term information. The LSTM mainly includes memory cell, an input gate, an output gate, and a forget gate. LSTM's have the capacity to remember their inputs for longer time. The key reason for this is the presence of a memory unit. This unit

helps to decide whether to write, remember or to delete the information from its memory.

2.2.1 LSTM Architecture

The main unit of LSTM cells are memory cell. Due to this reason the LSTM has outperformed RNN when there is the importance of the longer period of memory. Constant Error Carousel (CEC) units helps to solve the gradient vanishing problem present in LSTM[28].

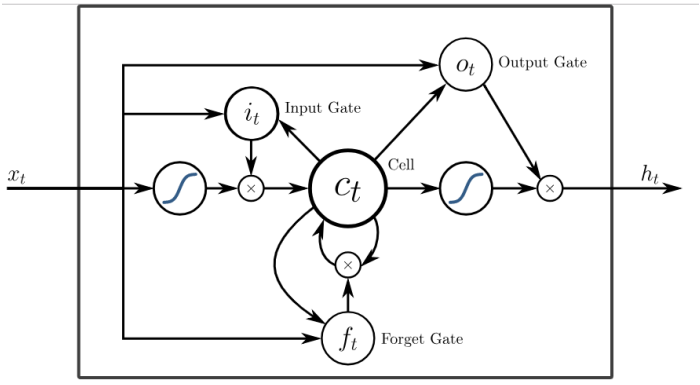


Fig. 4. LSTM unit with input (i), output (o) and forget (f) gates. Source: wiki

The activation function of the LSTM gates can be logistic function and tanh function. Now, ReLU function is also used majorly in all neural networks.

The gates are

1. Forget Gate - The forget gate helps to decide which information should be erased. Based on all previous state, LSTM cell decides the value either to be remembered or forget.
2. Input Gate - This gate is to update the cell state. To the sigmoid function, we insert input value and the output from previous state that is hidden layer output. Depending on the output value, the sigmoid output will decide which information is important to keep.
3. Output Gate - The output gate gives the output at the end of each state. These are the hidden states which contain information about the previous state.

Below diagram shows the architecture of LSTM

$$f_t = \sigma_g(W_f x_t + U_f c_t - 1 + b_f)$$

$$i_t = \sigma_g(W_i x_t + U_i c_t - 1 + b_i)$$

$$o_t = \sigma_g(W_o x_t + U_o c_t - 1 + b_o)$$

where x_t is the input vector to the LSTM unit f_t forget gate's activation vector
i_t input gate's activation vector
o_t outputs gate's activation vector
σ_g sigmoid function

2.2.2 Remarks on LSTM

1. Long Short Term Memory has overcome the disadvantage of RNN. Sometimes RNN gives the better result than LSTM.
2. Some applications of LSTM includes language modeling, speech recognition, image caption generation, time series anomaly detection, music composition, handwriting recognition
3. LSTMs are an extended version of RNN, where RNN do not work well for tasks where relevant events have more time steps. These have the vanishing gradient problems. LSTMs successfully removed this problem by introducing a memory cell which ensures the units to controlling information in and out of the cell. But LSTM had some problems related to forget cells. Later many changes had made which leads to peephole connections which enables LSTMs to learn precise intervals[28].
4. From the architecture of LSTM, we have known that LSTMs are difficult to train and optimize due to the need of tuning many parameters.
5. Gated recurrent unit (GRU) had proposed elements which has good advantage over LSTM due to its simplified structure.

2.3 GRU

A Gated recurrent unit (GRU) was introduced by Kyunghyun Cho et al.in 2014[23]. GRU is the Recurrent Neural Network and is similar to LSTM. It has only two gates, a reset gate, and an update gate.

Update Gate- Decides how much to keep the previous memory.

$$z_t = \sigma(W^z x_t + U^z h_(t-1) + b_z)$$

When x_t is included in the network, it is multiplied by the factor of its weight W^z. Similarly, $h_(t-1)$ which includes information about previous state and it is multiplied by the factor of its weightU(z). Both results are added together and a sigmoid activation is applied to have output between 0 and 1.

Reset gate- Combines the new input with previous memory.

$$r_t = \sigma(W^r x_t + U^r h_(t-1) + b_r)$$

where x_t input vector z_t update gate vector r_t reset gate vector W,U and b parameter matrices and vector GRU also solve the vanishing gradient problem found in standard RNN.

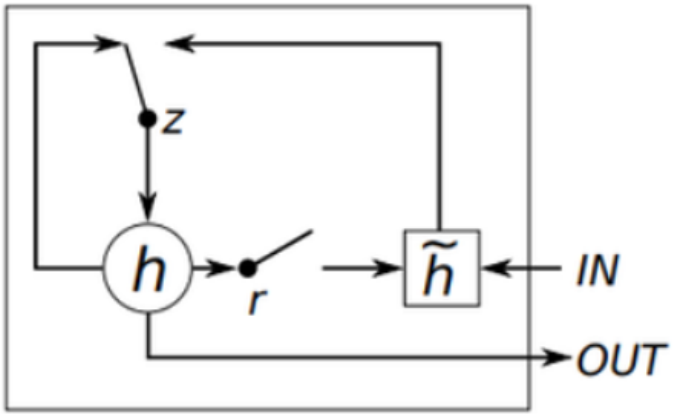


Fig. 5. Gated Recurrent Unit. Chung, Junyoung, et al. Empirical evaluation of gated recurrent neural networks on sequence modeling. (2014)

In Gated Recurrent Unit (GRU) the weights are updated by backpropagation through time. It has become very popular RNN due to its simple structure and that it has one gate less from LSTM[9].

From our research, GRUs train faster and GRUs are simpler due to the presence of only 2 gates compared to LSTM. Hence it is easier to modify the network. LSTM applies a nonlinearity (sigmoid) before the output gate, GRU shows that LSTM is more flexible compared to GRU.

2.4 CNN

In the 1950s and 1960s, the research was done by D.H Hubel and T.N Wiesel on the human's brain which lead to the idea of CNN[10]. The main advantage of Convolutional Neural Network is image recognition and classification. CNN are neural networks that use convolution layer in its architecture. The filters in CNN is moving across the entire input image, forms the dot product of input with weighting factor of the filters. Later, network learns from filters to detect specific features. In CNN, the network is introduced in such a way that, the input is considered as image. These then make the forward function more efficient by reducing the number of parameters. Then each input layers sequentially flows to convolution layers, pooling, fully connected layer, and finally to output softmax function[10].

The first convolutional neural network was the LeNet Architecture(1990s) introduced by Yann LeCun[11].

2.4.1 CNN Architecture

A convolutional neural network also has the normal structure as LSTM and input, output and hidden layers as GRU. The presence of convolution layer in the network stands out the CNN from other neural networks. Additionally, it has pooling and fully connected layers.

1. Convolutional layer mainly contains convolution operation. The input always passes through this operation. Later bias function will be added to the output. The convolution layers learn the model very deep by reducing parameters. In this way, it gives a solution to the vanishing or exploding gradient problems by using backpropagation[13].
2. Pooling Layers are also important components of the CNN. The purpose of a pooling layer is to perform dimensionality reduction of the input variables. The most common pooling function is the max-pooling function, which takes the maximum value of the local value. In average pooling, the average values is considered[26].
3. Fully-Connected Layer is a layer which connects every neuron from one layer to another .

LeNet-5 (1998)

LeCun et al introduced LeNet-5 in 1998. This is a seven layer convolution layer that helps for digit classification.

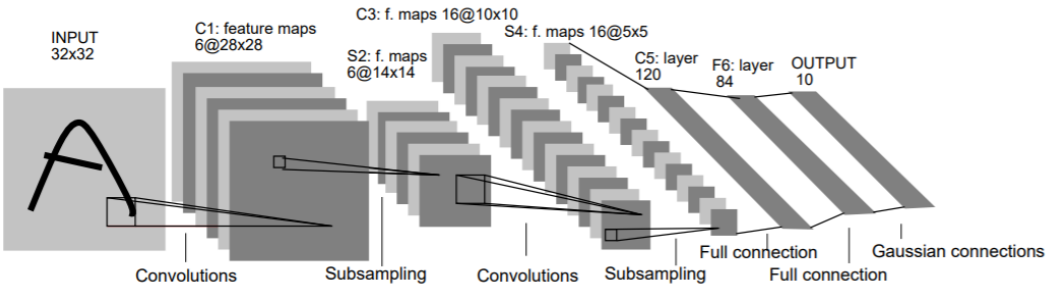


Fig. 6. Figure from [LeCun et al., 1998] Architecture of LeNet-5, a Convolution Neural Network, Source: Gradient-Based Learning Applied to Document Recognition Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner

In the first layer, the convolution layer converts the size of 32x32 into 28x28x6, so the input image is a pixel size of 32x32. The second layer in the LeNet-5 architecture includes the average pooling layer which helps to convert the image size to 14x14x6. In the third layer, it contains another convolution layer and the fourth layer is composed of average pooling layer. The fifth and sixth layers

contains fully connected layers. In the output layer it contains softmax layer which shows digits from 0 to 9 in the result[11].

The feature mapping, that is the output after applying the filters from previous filter for first convolution layer is 6. The image passed to second layer and third layer gives the feature maps of 6 and 16 respectively. In the architecture, the input values are normalized such that mean input is approximately equal to 0 and variance to 1. This helps to increase the learning rate.

Some applications include face recognition, image classification, action recognition, scene labeling.

2.5 Autoencoder

An autoencoder is a neural network helps to reduce dimensionality. Autoencoders are related to PCA but can learn more complex mappings due to their nonlinear nature

One of the important artificial neural network is autoencoder. The main function includes reconstructing the input after reducing the variables. Feature extraction algorithm plays an important role here. On the other side, it tries to get the input image from reduced encoding. An autoencoder consists of two parts: Encoder: This assists for feature extraction by reducing dimensions. It can be represented by an encoding function h=f(x). Decoder: This part is reverse to the encoder, where it reconstructs the input from reduced variables. It can be represented by a decoding function r=g(h)[16]. Choosing the loss function also plays an important role. There are two types of loss functions which includes mean squared error or cross entropy. The condition to choose cross entropy is when the inputs lie between 0 and 1. Types of Autoencoders are:

1. Denoising Autoencoder In denoising autoencoder, the input given is corrupted or the input added with noisy function. The main purpose of denoising autoencoder is the correctness of output regardless of its input and its flexibility to avoid the overfitting of the model.
2. Sparse Autoencoder In a normal autoencoder, every neuron of the hidden units get activated for each and every training example. But the sparse autoencoder focuses on mastering certain neurons for a certain type of training example. Hence, sparse encoder has a numerous neurons which are divided into groups.
3. Variational Autoencoder The variational autoencoder uses a latent representation which gives output as an additional loss component and deploy Stochastic Gradient Variational Bayes (SGVB) algorithm[15].
4. Contractive Autoencoder The contractive autoencoder focus on the robustness to the variations of input values, even the slight variations are considered. This is achieved by adding a regularizer to the objective function explicitly. This regularizer enables the function to be robust to the variations of input values.

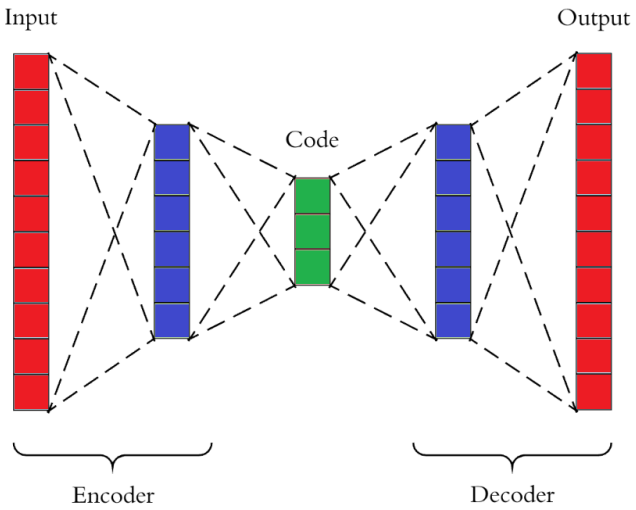


Fig. 7. The architecture of Autoencoder with Encoder, Decoder and latent representation

Fig 7 shows a more detailed visualization of an autoencoder. In the first part, the input goes to the encoder to attain the code. On the other part, it is in a reverse way. Here to achieve the output it uses only code. The final output should be same as input given[15]. In between encoder and decoder process, code where the dimensionality of variables after reduced.

Before training, the three aspects needed to be considered are:

1. Size of code
2. Number of nodes per layer
3. Loss function

2.5.1 Remarks on Autoencoder

Relationship with PCA: Autoencoders are similar to PCA but autoencoders are more flexible due to its structure. PCA can perform only linear transformation whereas autoencoder can perform both linear and non-linear transformations[29]. Hence it is clear that autoencoders are more advantageous due to its deep learning network. An autoencoder tries to gather more information about the given input but not the important or required information. Hence, sometimes it is tedious to model an autoencoder. Encoding layers try to preserve quantity of information rather than the quality of information. Autoencoders can be a good choice with unsupervised data.

3 TensorFlow and Keras

3.1 TensorFlow

TensorFlow was first developed by Google's AI organization. Numerous researchers and engineers from the Google Brain team involved in the project. This has been a very strong and flexible tool to develop machine learning and deep learning models. This has been an open source software library for complex numerical computations across a variety of platforms like CPUs, GPUs and TPUs[17]. A simple TensorFlow code to implement a multiplication between two constants is included in Appendix-1

Some of the main commands of TensorFlow used in our studies:

1. tf.Graph - In order to visualise computational graph, this command must be used.
2. tf.Session - In order to enable the computations this graph is used.
3. tf.Operation - It includes calculations.
4. tf.Tensor - These represent the numerical values in the form of tesors.
5. tf.layers.Dense layer- It processes the batch of input vectors to give out a single output.
6. global_variables_initializer - helps to initialize every variable declared.
7. tf.losses - enables loss functions.
8. tf.train.Optimizer - enables optimizing function.
9. tf.train.GradientDescentOptimizer - It enables the gradient descent optimizer as an optimizing function.
10. tf.train.AdamOptimizer - It enables the Adam optimizer.

3.2 Keras

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation.

Keras is built on the top of TensorFlow, CNTK or Theano. In our experiments, TensorFlow runs in the backend of Keras. Keras is simpler to use than TensorFlow and enables fast implementation. Though the Keras is less flexible[19].

Keras is more efficient to use when easy and fast prototyping is important. It has an extensive function to support recurrent neural networks as well as convolutional neural network. Like TensorFlow, Keras also has features like activation function, optimizers, and objectives. Some of the main advantages of using Keras are user-friendly, modular and composable and easy to extend[20].

1. tf.keras.Sequential model- helps in stacking different functions to plain model.
2. tf.keras.layers help to set activation function and also regularization.
3. tf.keras.Model.compile enables three funcions which includes optimizing function, loss function (tf.keras.losses) and metrics (tf.keras.metrics).

4 Study and Implementation of MNIST dataset

The MNIST data set contains 55000 training images, 10000 test images and, 5000 validation images. These images correspond to the digits from 0 to 9. Each digit information is stored in 28 28 which corresponds to 784 pixels.

The MNIST data is directly accessible through TensorFlow. Each image can be seen as 28 rows of 28 pixels (28 x 28). To run one complete image, the timesteps taken here is 28. In other words for each time step, one row of 28 pixels are fed into the LSTM unit. The batch sizes chosen here is 128. So in order to complete the 128 images, 28 timesteps are taken. The first row of all 128 images are fed in every single timestamp. The input arguments of LSTM accepts tensors of shape [batch_size, input_size]. The input argument from [batch_size, input_size] is transformed into [batch_size, num_units] in the output. We followed this[21] tutorial to know more about the usage of LSTM in TensorFlow.

4.1 LSTM Implementation

In the implementation we used the time_step of 28, num_units is 128. Firstly we have declared the weights, bias variables and placeholders. Secondly we need to modify the shape of the output format to (batch_size, n_classes). Now in order to give input to static_rnn, we need to unstack the input from (batch_size, time_steps,n_inputs) into number of time_steps x(batch_size,n_input). The algorithm for implementation are as follows:

1. The training set consists 55000 images 28x28 pixels each
2. Initializing constants like time steps, hidden LSTM units, batch size.
3. Declare placeholders, weights, and bias variables.
4. Defining the network
5. Generate the prediction
6. Define loss, optimizer, and accuracy
7. Calculate test accuracy

Now we need to define the model by using basiclstmcell and we can predict the output. By defining loss function, optimizer we can make the model evaluation. The error between real and predicted image can be found by loss function. Common loss functions are mean squared error and cross-entropy.

4.2 Our Contribution

Comparing different optimizers. Here we are comparing two optimizers namely Adam Optimizer and Gradient Descent Optimizer to check the accuracy. By doing all the steps we received the accuracy of 99.32%. Below image shows the results. Implementation can be seen in Appendix-2

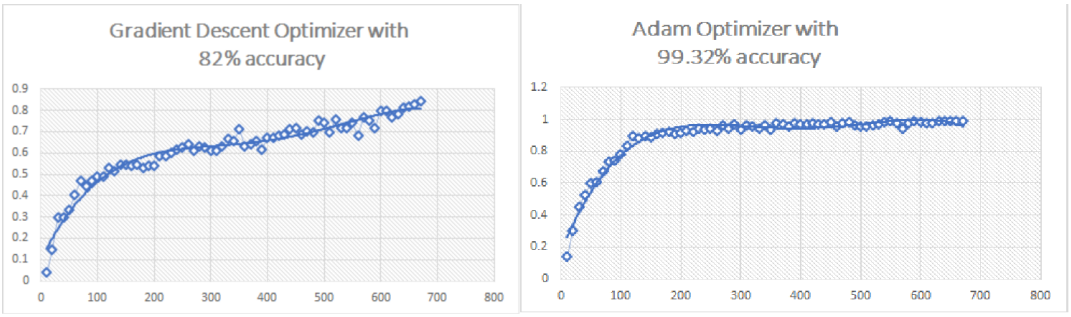


Fig. 8. Results showing the mnist dataset where Adam Optimizer shows 99.32% of accuracy and Gradient descent optimizer shows less accuracy of 82% accuracy after tuning.

From the results ,

1. we can conclude that Adam Optimizer gave good results with less error rate compared to Gradient Descent optimizer. Adam algorithm controls the learning rate. Adam combines advantages of AdaGrad(Adaptive Gradient Algorithm) and RMSProp(Root Mean Square Propagation)- maintains the learning rate and the algorithm finds an exponential moving average of the gradient and the squared gradient[31]. Hence Adam Optimizer is better.
2. Some experiments were done to check the accuracy, where batch size is decreased that gave the better result with long processing time
3. When epochs are given more, it gave better accuracy
4. When epochs are very high, the training accuracy was good but the prediction gave high loss values with less accuracy due to overfitting

During running the codes in the jupyter notebook, we have come across many errors. In that one of them is "Attempted to use a closed Session". This error occurs when we didn't reset the graph or if we use the same graph multiple times. The solution is using the command tf.reset_default_graph() after declaring the input to TensorFlow.

Another error we come across while during experiments was "Variable rnn/basic_lstm_cell/kernel already exists, disallowed". The solution was to indent a line by 4 spaces or Creating session without block and close it manually.

4.3 Autoencoder Implementation

As we have mentioned earlier, here also we are dealing with mnist dataset which are 28x28 pixels i.e. 784 pixels with autoencoder. We have mentioned in the previous section that autoencoder consists mainly of 2 parts. The encoder which has an input layer of 28x28 i.e 784 dimensions and some hidden layers. And in the decoder which is reverse, that is the hidden layers help to go up to the size of the input layer. We add Variance_scaling_initializer() is the weight initializer

which works with ReLU activation function In the encoder network we define the number of hidden layers unit and add dropout layer. In the decoder layer, the difference is that no dropouts are given. Mean squared error is used as loss function. As we learned from our previous implementation, Adam Optimizer is better, we have used the same. This[18] tutorial is followed to understand the working of autoencoder. Here we have compressed the images from 784 pixels into 196 pixels. And our accuracy is high. This shows the importance of autoencoder in reducing dimensionality. Code can be seen in Appendix-3

4.4 Our Contribution

Initially, we did some experiments by considering epoch values. When the epoch value is given is 5, then we got the loss function as 0.0230

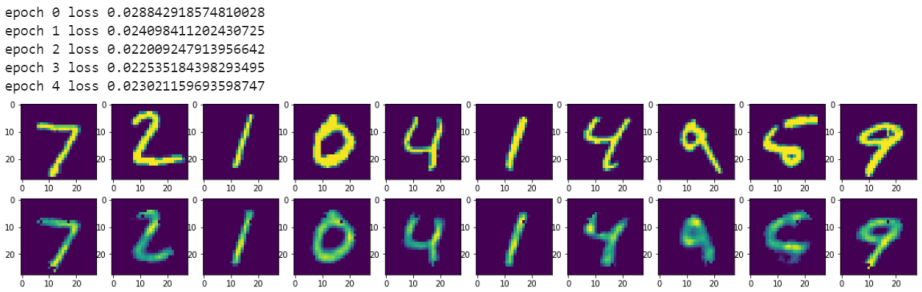


Fig. 9. The result of the mnist dataset with Autoencoder with epoch 5 shows error of 0.0230

As we increased epoch to 15, the improvements in the results are slightly high. But when we increased epoch to very high, the results don't show many improvements. This can be due to the overfitting

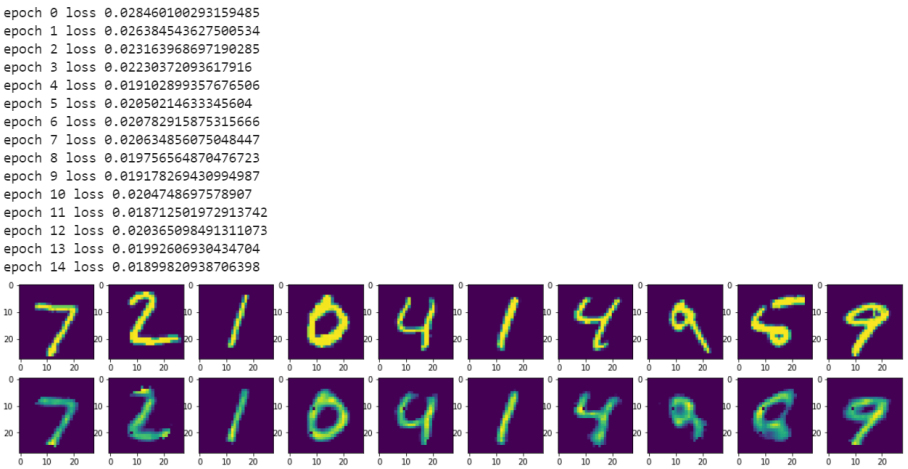


Fig. 10. The result of mnist dataset with Autoencoder with epoch 15 and batch_size 150 shows error of 0.016

In this part, we did some analysis by changing the batch_size . When the batch_size was 150, we got the loss function as 0.0189, as we increased the batch_size to 200, the loss function increased to 0.0196.

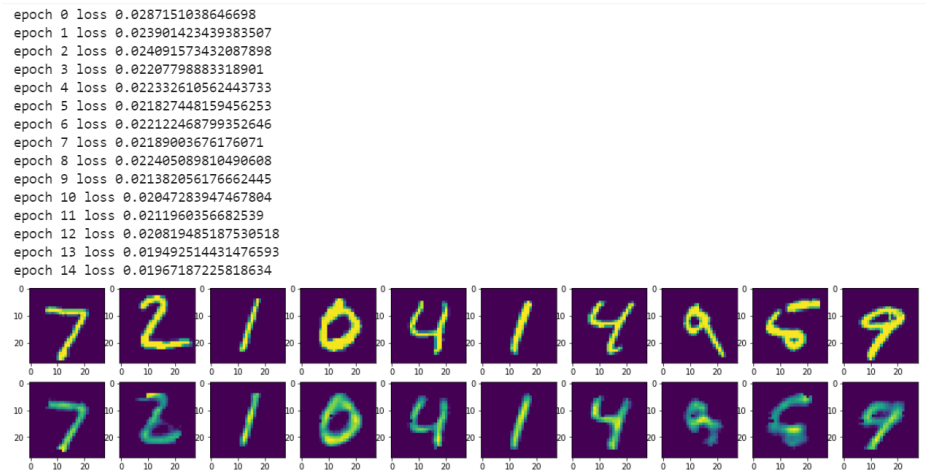


Fig. 11. The result of mnist dataset with Autoencoder with batch_size 200 shows error of 0.016

Experiments done on the feature compression. As the number of the compression values are decreased, more information is lost. Hence the loss function gradually increased can be viwed from the table 1.

Table 1. Experiments done on different feature compression values

Experiments	Feature Compression	Epoch 0	Epoch 1	Epoch 2	Epoch 3	Epoch 4
1	196	0.0280	0.0242	0.0220	0.0225	0.0230
2	100	0.0380	0.0419	0.0386	0.0344	0.0385
3	50	0.0439	0.0416	0.0399	0.0404	0.0401

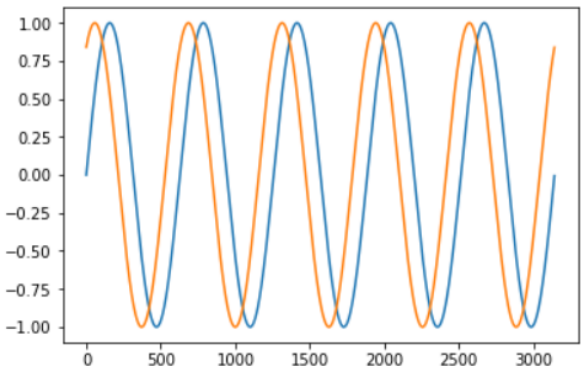
5 Implementation of Time Series Prediction of Sine wave

Basic LSTM cell:

The basic LSTM cell we used in this demonstration is tf.contrib.rnn.BasicLSTMCell (n_hidden). This is a cell unit of basic LSTM recurrent network cell in TensorFlow. The n_hidden should be assigned to the number of LSTM units to be used. This cell creates a LSTM layer and initiates all the required variables for all gates.

5.1 Implementation of Time Series Prediction of Sine wave using basic LSTM

For this sine wave implementation, let us consider two sine waves namely x and y. y is nothing but the shifted signal of x. At first, the two sine waves are generated. And then the x signal is shifted for a time window of size say 5. The LSTM units are then given time_steps, equal to the time window size. Here in this basic implementation batch size allowed is equal to only one. Then the LSTM model is trained with shifted x and y. Here the output is then considered as the output of LSTMs, that of from the training input, so simply the output is considered to be the trained output itself rather than that for test data. This example is considered for a better understanding of how a training model is built and how the LSTM works for this simple example .We followed this [26] tutorial to implement this model. Using different optimizers led to the same result approximately. The Gradient descent and Adam optimizer both performed similarly, leading to satisfactory results. The input and output graphs are shown below. The implementing code can be seen in Appendix - 4




```
[ 0.          0.00999983  0.01999867 ... -0.02592363 -0.01592586
 -0.0059265 ]
```

Fig. 12. The Sine waveform of x and y

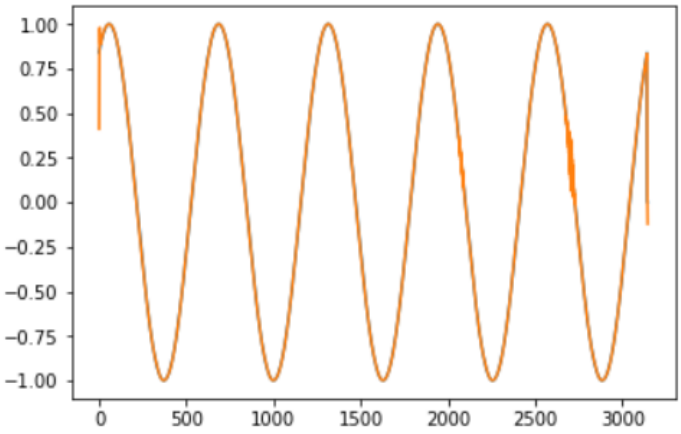


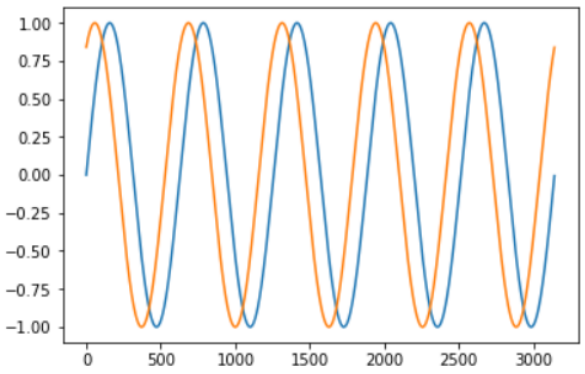
Fig. 13. Output wave of LSTM and y plotted together

Our Contribution

5.2 Further developing the flexibility of the code (Appendix - 6)

The previous example was demonstrated with many restrictions like with only batch size of 1 and no function to implement to pick batch size elements, also no future predictions. In this implementation, we are expanding the code of flexible batch size. And to perform this a function is also employed. But as a result of this, there is a compromise in the quality of the output. That is as the batch size increases the output deteriorates. For the smaller batch size, the performance is

good. Here surprisingly the Gradient descent optimizer performed comparatively well over Adam optimizer for a certain number of runs. The window size or time_steps did not affect the result in the final.




```
[ 0.          0.00999983  0.01999867 ... -0.02592363 -0.01592586
 -0.0059265 ]
```

Fig. 14. The Sine waveform of x and y

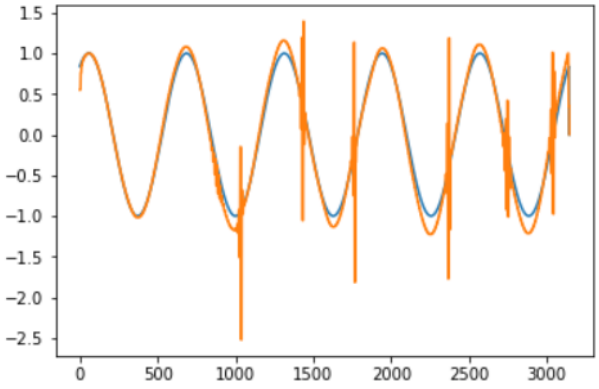


Fig. 15. Output wave of LSTM and y plotted together

5.3 Implementation with GRU (Appendix-5)

Here we have used the previous implementation of sine wave prediction. We changed the cell type tf.contrib.rnn.BasicLSTMCell (n_hidden) to tf.contrib.rnn.BasicGRUCell (n_hidden), just to compare the output result between LSTM and GRU. Every

parameter settings between the two examples were the same other than the basic cells. The LSTM performed better over GRU in this particular example. It may be due to its complex structure.

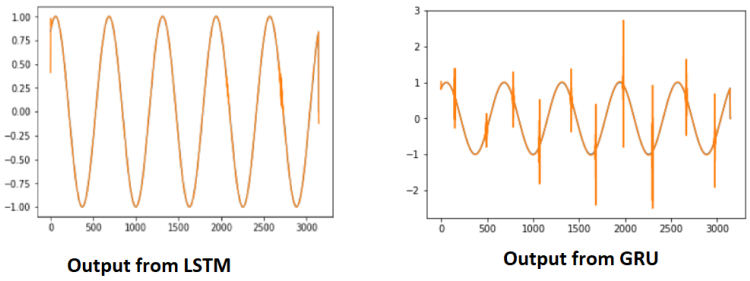


Fig. 16. Output wave of LSTM and y plotted together

5.4 Implementation with Keras (Appendix -7)

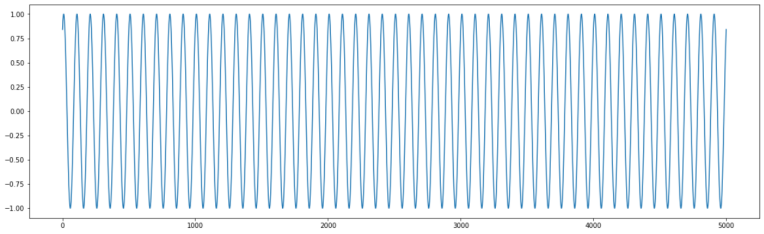


Fig. 17. Sine wave input

Keras is a platform which runs over TensorFlow useful for building Neural networks. The previous development is now extended with help of Keras, which results in better accuracy and simple implementation without much complications. This Keras model is able to predict one step ahead of the sine wave giving window size of input say 50. One more important aspect is that the LSTM model is stacked here. Keras allows stacking multiple layers easily just by the command .add . Here firstly the sine wave csv file is imported. For a good model building, we scale the values between -1 to 1. Then the sine wave is shifted by a certain time window. Then the last column is set as the output to be trained. Then the Keras model is built with linear activation and MSE as the parameter for error. The optimizer used here is Adam Optimizer. The data is divided into 80% as training data, the rest 20% is test data. The model is trained with the training

data, and then test data is given to predict. The model gave a satisfactory output with many more accuracy. So Keras was helpful for simpler implementation and also with notable results.

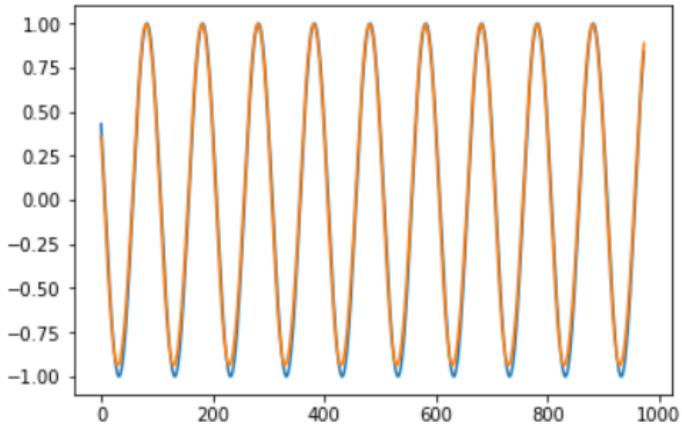


Fig. 18. Output wave of LSTM and y plotted together

5.5 Further development with Keras (Appendix -8)

In this implementation, the sine wave is predicted with multiple shifted sine waves. So multiple shifts are predicted here

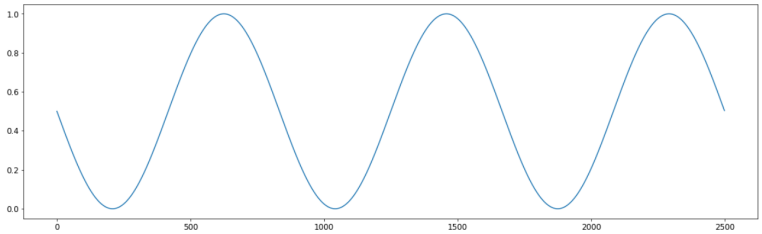


Fig. 19. Input sine wave

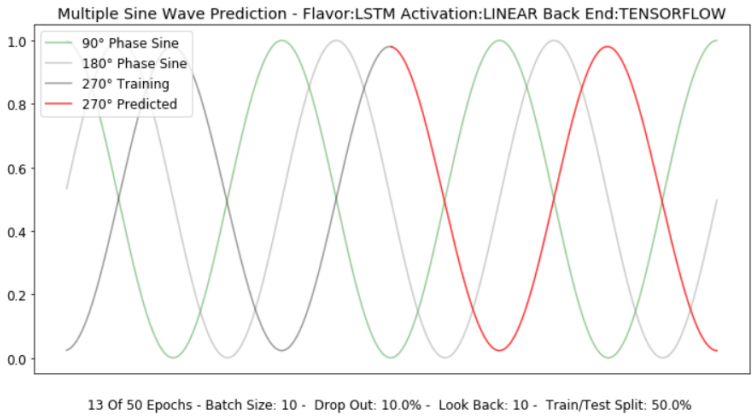


Fig. 20. Shifted Output wave of sine wave with 90,180 and 270 degree shift

6 LSTM for Anomaly Detection in Time Series Paper

In this[22] paper the stacked LSTM networks are used for anomaly or fault detection in time series. The network is trained for non anomalous data. In other words, the model learns the normal behavior. Then anomalous data along with normal data is given for validation(For Multivariate distribution) and test set. The resulting prediction errors are modeled as a multivariate Gaussian distribution. A threshold is set to the log values of the probability density of errors. Later it is used to check the likelihood of anomalous behavior. Based on this information the test data is predicted whether it is an anomaly or normal.

Stacked LSTM is defined as LSTM model comprised of multiple LSTM layers. It provides an output at each time step rather than one output for all input steps. The main advantage of using stacked LSTM is to learn a hierarchical representation of time-series data and it also enables more complex representation of time-series data. In this paper, normal data is divided into Normal train, Normal Validation 1, Normal Validation 2 and Normal Test data. Anomalous sequences are divided into Anomalous Validation and Anomalous Test. Firstly, training the network with the normal test inorder to learn the normal sequence. Normal validation 1 is used for early stopping while learning the network weights. The model which is trained in normal training data used to find error vectors. These vectors are used to fit Gaussian distribution. Later Normal validation-2 and Anomalous validation are used to learn threshold value. Finally test sets used for the final prediction of the network where above the predefined threshold value is considered as anomalous. Experiments are done on 4 different data sets which are ECG, Space Shuttle, Power data, and Multi-sensor engine. The analysis is done with Precision, Recall, and F-score value. Here they have used as 0.1 score which are giving 10 times the importance to precision which tells that there should not lose any anomalies, but they can find false anomalies.

Finally, they have combined the power of LSTM with recurrent hierarchical processing layers for predicting time series and used it for anomaly detection. The results showed that Stacked LSTM networks can learn higher level patterns without prior knowledge. From observations, they claimed that during long term dependencies LSTM gave more robust result than RNN otherwise, both gave similar results.

The prediction error is modeled as multivariate gaussian distribution. The probability distribution of the errors made while predicting normal data is obtained. These are used to analyze the likelihood of the anomalous data

6.1 Critical Points

The main critical points are:

1. The main confusion occurred in understanding the paper was the term used "RNN", which they didn't explain properly.There is a number of variants of RNN. But they did not mention which variant of RNN was used for comparison with LSTM. This method mentioned in the paper is not in detail.
2. Terms used for analysis like Precision, Recall and F-score value can be criticised. Since 0.1 F score value has been used which gives more importance to precision on the other hand, it is advantageous as well.
3. "l" which is future prediction values are not specified properly which is one of the important values are needed to be specified.
4. In the paper, they have mentioned traditional methods like CUSUM, EWMA. But they are unable to compare results with these methods to see the benefits of deep neural networks.

6.2 Remarks on Paper

Terms used for analysis like Precision, Recall[14] and F-score value are not so appropriate according to our knowledge.

1. Precision - It is the ratio of correctly predicted positive values to the total predicted positive values.

$$Precision = TruePositive/(TruePositive + FalsePositive)$$

2. Recall - Recall (Sensitivity) - The ratio of correctly predicted positive values to the all values in actual class.

$$Recall = TruePositive/(TruePositive + FalseNegative)$$

3. F-_beta value-F_beta Score is the weighted average of Precision and Recall. The fofrmula shows it uses true positive, false positive and false negative.

$$F_\beta = (1+\beta^2)TruePositive/((1+\beta^2)TruePositive+\beta^2 FalseNegative+FalsePositive)$$

Table 2. Summary of 8 variables with Statistical measure like Features, NA's, minimum, 1st quartile, median, mean, 3rd quartile and maximum value which gives an brief overview of variable information

Experiments	True Positive	False Positive	True Negative	False Negative	Precision	Recall	F_0.1 score	Balanced Youden Index
1	35	2	38	5	0.94	0.87	0.93	0.87
2	40	0	40	0	1	1	1	1
3	40	0	0	40	1	0.5	0.99	0.5
4	20	20	20	20	0.5	0.5	0.5	0.5

In order to analyse the results of F0.1 value (which gives 10 times more importance to Precision than Recall) for various set data we made some experiments. From above table experiment no 3 shows that even though the results are not very good, F0.1 value gave 0.99 which shows that it doesn't give importance to Recall. Then we made an analysis with other parameter Balance Youden Index which gave a more better result which considers a minimum of Precision and Recall. But in anomaly detection always Precision is important than Recall. Hence we cannot conclude any one parameter is best.
GRU combines both input and forget gates into single update gate. Due to this simplified architecture GRU is computationally more efficient than LSTM. Hence comparing the results between LSTM and GRU would have been a popular alternative instead of RNN[24] and this is considered in the future work of our paper.

6.3 Implementation (Appenix-9)

Here we have tried to implement the LSTM for Anomaly Detection in time series. In the implementation, we used power data set.

6.3.1 Power Data set The data set is taken from this[27] website. The explanation on this data set is given here [31]. The data set is related to power consumption. The power consumption on the weekdays are expected to be high and that on the week end is expected to be low.This is shown in Fig 21. The reading is taken for every quarter hour for one year. 672 observations are made for one week. Totally there are 35040 observations of power consumption. If the power consumption on week day is low or the power consumption on week end is high ,it is considered to be anomaly. This is hown in Fig 22 We divided the data such that Normal data consisted of 34 weeks and Normal Validation set consists of 3 weeks. The Anomaly validation has 8 weeks and test set has 7 weeks. There were a total of eight anomalies with seven anomalies corresponding to weekdays with low demand and only one weekend as an anomaly with high demand. Out of the eight anomalies, four were allotted to validation and the remaining to test data.

1. The normal data set for training(Fig 24)
2. The normal data set for validation-1(Fig 25)
3. The normal data set with 3 Anomalous values for validation-2(Fig 26)

4. Both normal and anomalous data set for testing (Fig 27)

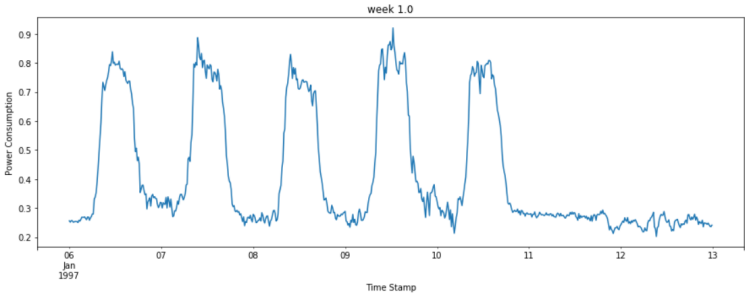


Fig. 21. Visualization of Normal Data in 1 week

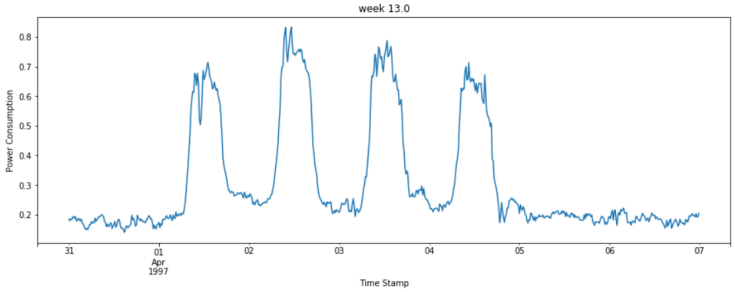


Fig. 22. Visualization of Anomaly Data in 1 week

After the data set is ready, we implemented a model with LSTM neural network. By using the normal data set, we trained the model with 20 time window size and to predict the next value(one step). Next we plotted the error vector from the errors obtained in the model implementation. Later we modeled our error vector into multivariate gaussian distribution. From this we calculated mean and variance. In the next steps, we applied log pd value, through this we are able to know the probability of error in the data set. Now for our LSTM model, we are applying validation set -2 which includes both normal data and anomaly data. The error vector obtained from the predicted value is modeled as gaussian distribution and log pd value applied. Now from this value, we set the threshold value. Finally, we are using testing data to test the model, here by using threshold value we are able to detect the anomaly.

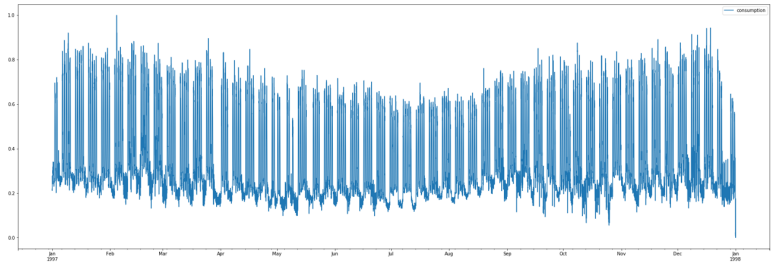


Fig. 23. Time Series of Normal Data

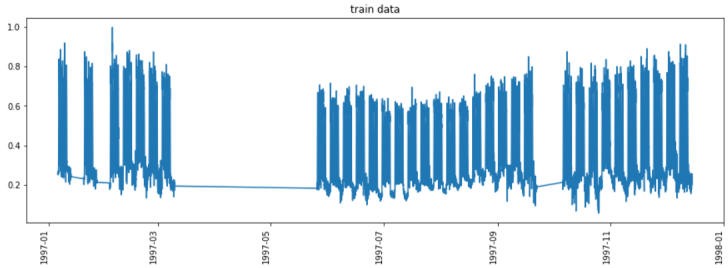


Fig. 24. Training data in TimeSeries

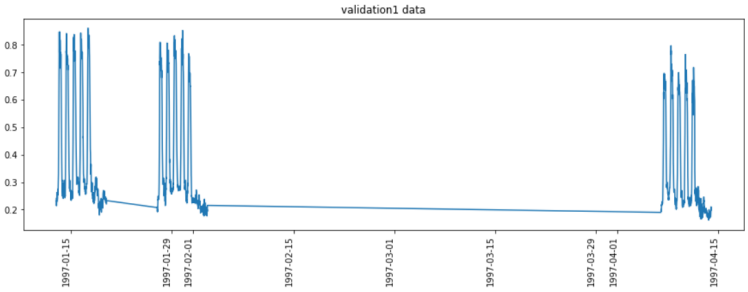


Fig. 25. Validation data-1 in Time Series which has normal data in blue

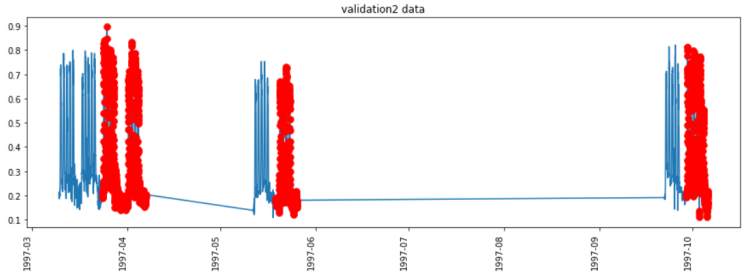


Fig. 26. Validatin data-2 which has normal data in blue and anomaly data in red

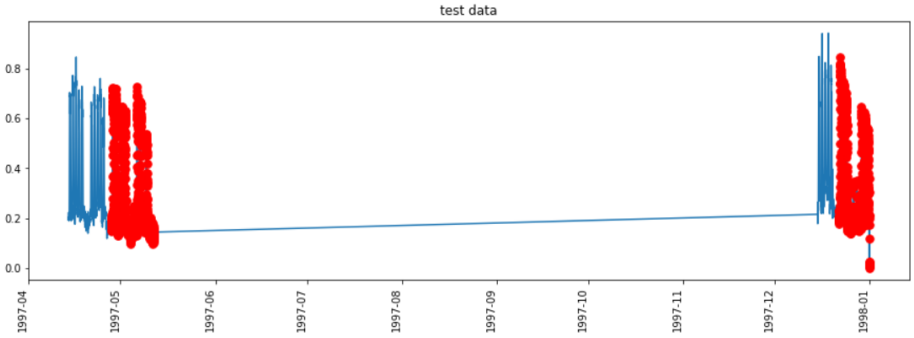


Fig. 27. Test Data in TimeSeries

The error vectors that is obtained from training data set is plotted by superimposing the training data with predicted data and finding the difference between them. This is hown in Fig 28.

Fig. 28. Error vectors from training data

For the error vectors mean and covariance is found and multivariate normal distribution is applied to find probability density of errors . This is shown in Fig 29.

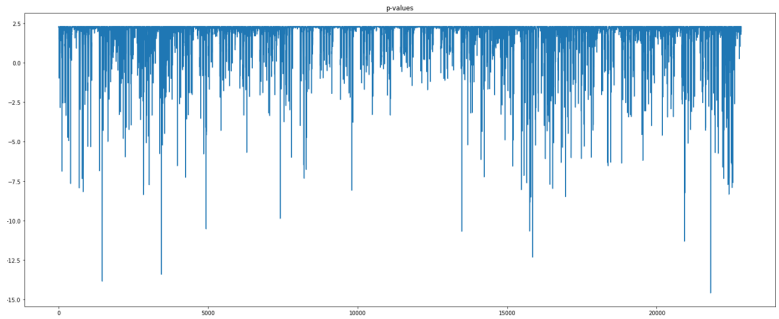


Fig. 29. Probability density of training error vectors

Then by trial and error method the threshold is set and anomaly lines are plotted for the points where probability density is low. But the model could not identify the true anomaly in this case. For few runs initially it was able to detect few anomalies. This is shown in Fig 30.

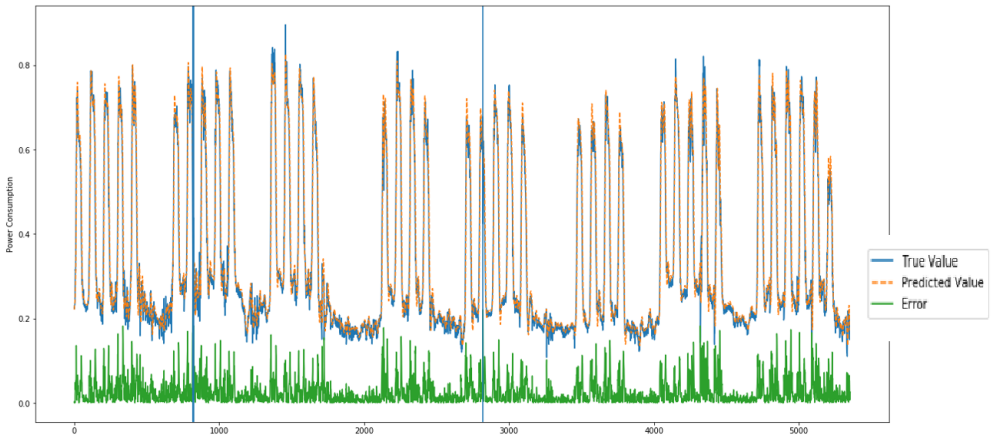


Fig. 30. Anomaly lines for validation 2 data set

The points that are crossing the threshold is considered to be anomaly in Fig 31 but the anomalies are falsely detected.

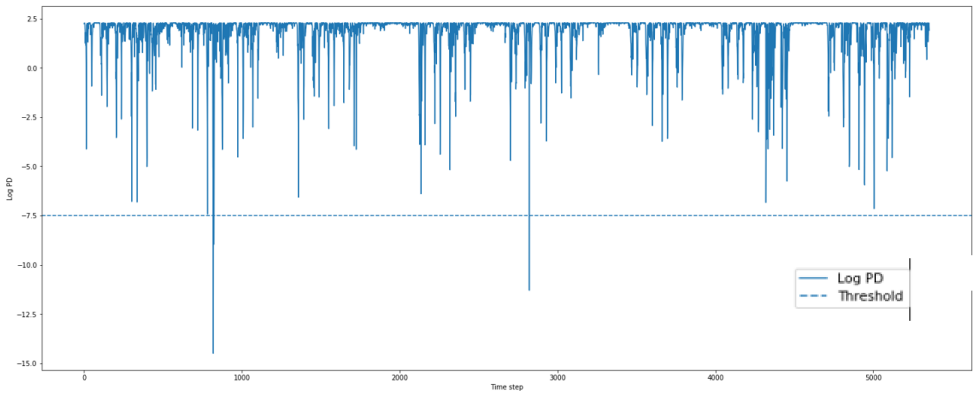


Fig. 31. Probability density of error vectors for validation 2 with threshold

With the given threshold anomaly lines for the test data is shown in Fig 32. Again the model did not detect true anomalies. The probability density of error vectors for test data is shown in Fig 33.

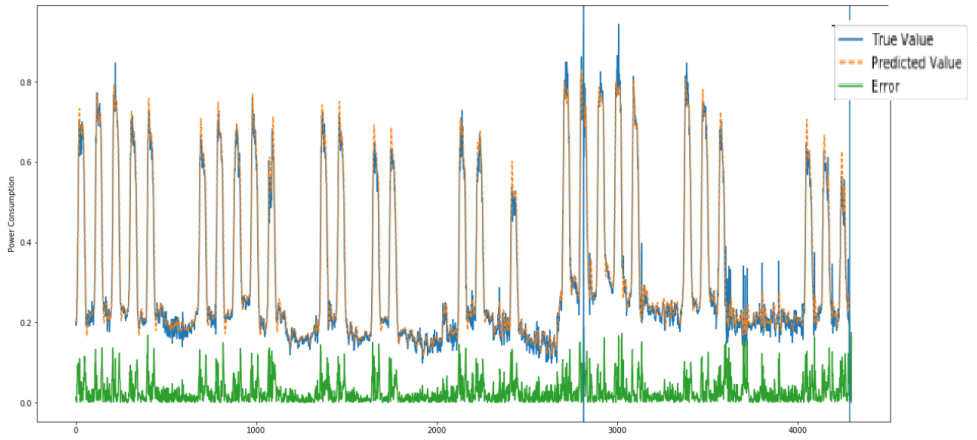


Fig. 32. Anomaly lines for test data set

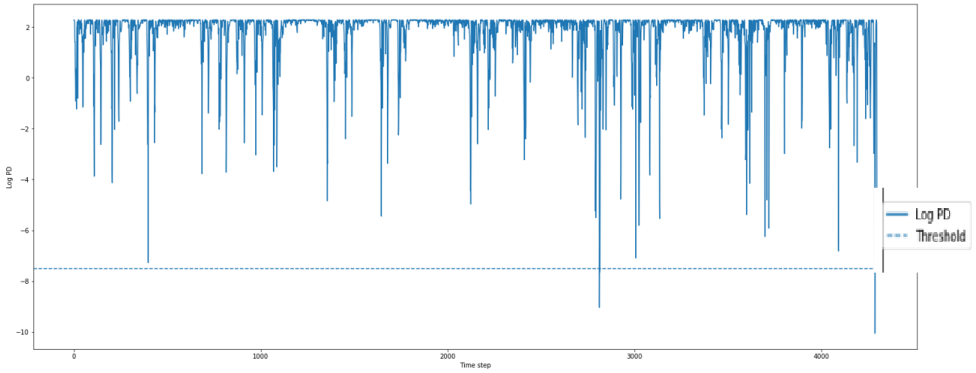


Fig. 33. Probability density of error vectors for test with threshold

6.4 Implementation to predict power consumption one day in advance:

In this implementation, the model is trained to predict the power consumption for the next day. That is the time horizon is 96th point after the training end point. Carrying the similar steps as previous experiment and when the threshold was set to -6, the anomalies in the week of 12th and 39th week were detected successfully. But failed to detect the anomaly in the 20th week and 13th week. This is shown in Fig 34 and 35.

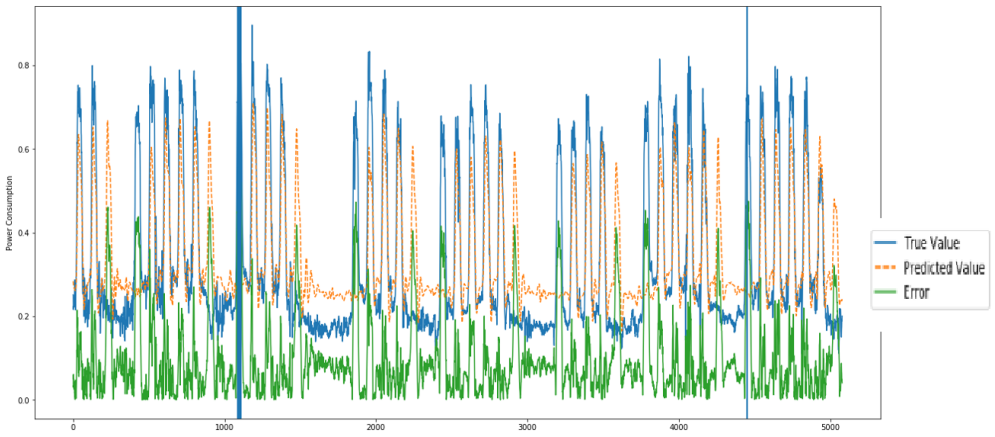


Fig. 34. Anomaly lines for validation 2 data set

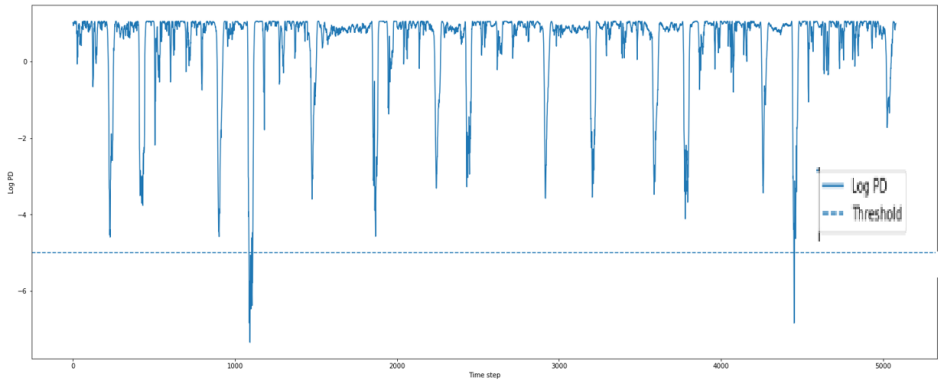


Fig. 35. Probability density of error vectors for validation 2 with threshold

In test data the anomaly was detected in 18th and 51st week. But failed to detect anomaly in 52nd and 17th week. This is shown in Fig 36 and 37. So the anomaly detection was effective when data was predicted one day ahead than predicting 15 mins ahead. This may be due to the fact that LSTM learns better to predict shorter time steps in future and the chances of big errors are less.

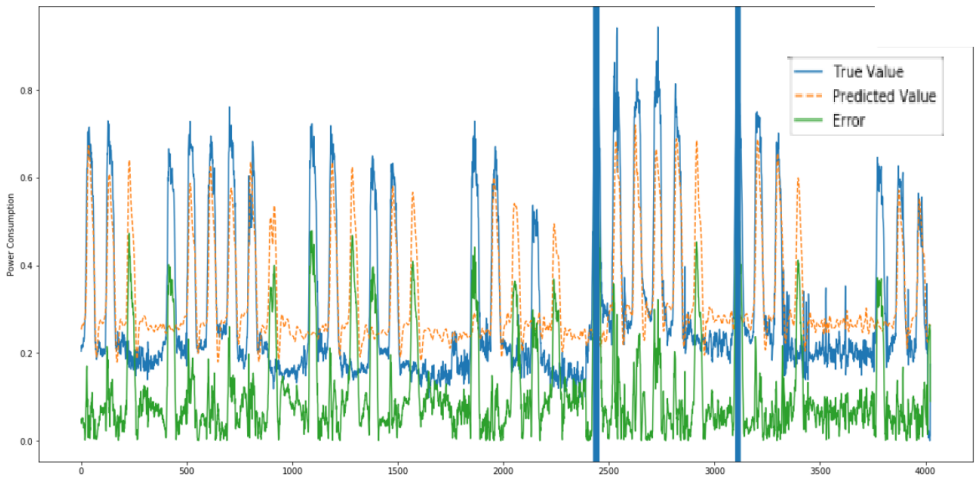


Fig. 36. Anomaly lines for test data set

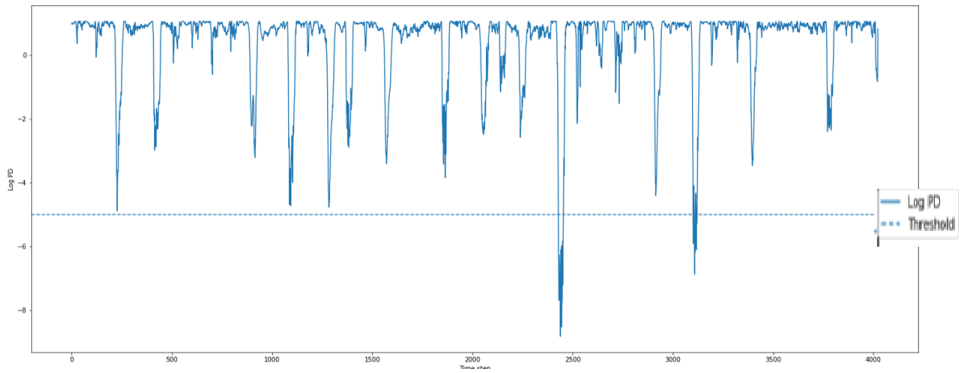


Fig. 37. Probability density of error vectors for test with threshold

7 Conclusion

In this project, we learnt the state-of-the-art deep learning algorithms like RNN, LSTM, GRU, autoencoder, and CNN. In our experiment of sine wave prediction, LSTM gave better result than GRU but as that was a simple implementation, GRU was expected to perform as equivalent to LSTM so the model parameter tuning could have achieved this. This can be considered as feature scope of the project. When shifted from TensorFlow to Keras, we found Keras is very flexible and code implementation seems to be easier than TensorFlow. In tenorflow picking the next batches was implemented as function but the keras does it on its own. Our experiments with optimizers showed that Adam Optimizer gave better result than gradient descent optimizer due to the self learning of adam optimizer. From some experiments we got to know that smaller the batch size better the result. Increasing time window size helps the model to learn better in sine wave prediction. An experimient was performed to produce the phase shifted sine wave. Common errors found in TensorFlow was discussed in the above sections. Initially we found difficulty, when fitting the data into the LSTM shape. In autoencoder we did experiments by varying epochs and the feature compression. Lesser the feature compression gave higher the value of loss function due to the the information loss.

The Paper reading gave us insight to understand the scientific paper, to mark few critical points on the paper and also to detect anomaly in the time series using stacked LSTM ntworks. In the paper "Long Short Term Memory for Anomaly Detection in Time Series", some important things like future steps(Horizon) and "RNN" term was not clearly explained. And inorder to implement the paper numerous things were made as assumption. As we mentioned earlier, comparing the result with GRU would be the better idea, this is also considered as future scope of the project. The LSTM implementation of this paper was made, and the model which predicts for one day ahead was able to detect more anomalies than the model which predicts for just 15 mins ahead. This could be a evidence for

LSTM being a effective predictor for the time series data and also it's capability in predicting anomalies.

8 Appendix

These are included in SVN folder 2018-19_CaseStudy

1. Tensorflow_multiply
2. Tensorflow_MNIST
3. Tensorflow_Autoencoder
4. Tensorflow_sineLSTM
5. Tensorflow_sineGRU
6. Tensoeflow_sineLSTM2
7. Keras_sineLSTM
8. Keras_sineLSTM2
9. Powerconsumption

References

1. Jrgen Schmidhuber (2015). Deep Learning. Scholarpedia
2. Yann L., Yoshua B. & Geoffrey H. (2015) Deep Learning
3. Nikolaus Kriegeskorte (2015) Deep neural networks: a new framework for modeling biological vision and brain information processing
4. Schmidhuber, J. (2015). "Deep Learning in Neural Networks: An Overview". Neural Networks.
5. Elliot, David L. (1993), "A better activation function for artificial neural networks", ISR Technical Report TR 93-8, University of Maryland, College Park, MD 20742.
6. Williams, Ronald J.; Hinton, Geoffrey E.; Rumelhart, David E. (1986). "Learning representations by back-propagating errors"
7. Sepp Hochreiter; Jrgen Schmidhuber (1997) "Long short-term memory". Neural Computation.
8. Felix A. Gers, Nicol N. Schraudolph, Jurgen Schmidhuber (2002) Learning Precise Timing with LSTM Recurrent Networks.
9. Rahul Dey and Fathi M. Salem (2017) Gate-Variants of Gated Recurrent Unit (GRU) Neural Networks.
10. Wurtz, Robert H.(2009) Recounting the impact of Hubel and Wiesel .
11. Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner(1998) Gradient-Based Learning Applied to Document Recognition .
12. Filippo Maria Bianchi1a, Enrico Maiorinob , Michael C. Kampffmeyera, Antonello Rizzib, Robert Jenssena (2017)An overview and comparative analysis of Recurrent Neural Networks for Short Term Load Forecasting
13. Spiros V. Georgakopoulos, Sotiris K. Tasoulis, Aristidis G. Vrahatis, Vassilis P. Plagianakos (2018) Convolutional Neural Networks for Toxic Comment Classification
14. Jesse Davis, Mark Goadrich(2015) The Relationship Between Precision-Recall and ROC Curves

15. Chen, Min & Shi, Xiaobo & Zhang, Yin & Wu, Di & Guizani, Mohsen. (2017). Deep Features Learning for Medical Image Analysis with Convolutional Autoencoder Neural Network. IEEE Transactions on Big Data. PP. 1-1. 10.1109/TBDATA.2017.2717439.
16. D. Charte, F. Charte, S. Garca, M. J. del Jesus, and F. Herrera(2017)A practical tutorial on autoencoders for nonlinear feature fusion: Taxonomy, models, software and guidelines,
17. https://www.tensorflow.org/ Retreived: 24-12-2018
18. https://towardsdatascience.com/deep-autoencoders-using-tensorflow-c68f075fd1a3 Retrieved: 24-11-2018
19. "Keras Documentation"https://keras.io/ Retrieved: 04-01-2019
20. "Core - Keras Documentation" https://keras.io/layers/core/ Retrieved: 02-01-2019
21. Yosri LAifi (2018) Best explanation of Recurrent Neural Network (LSTM) with TensorFlow & MNIST dataset Retrieved:29.10.2018
22. P. Malhotra, L. Vig, G. Shro, and P. Agarwal, (2015)Long Short Term Memory Networks for Anomaly Detection in Time Series, in Proceedings.
23. J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, (2014)Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling,
24. K. Greff, R. K. Srivastava, J. Koutnk, B. R. Steunebrink, and J. Schmidhuber,(2017) LSTM: A Search Space Odyssey,
25. Jozefowicz, R., Zaremba, W., and Sutskever, I.(2015) An Empirical Exploration of Recurrent Network Architectures
26. https://drive.google.com/file/d/15wl3ugPAShnaHWsl_XXPrU3p1ZQ4R4oj/view
27. http://www.cs.ucr.edu/ eamonn/discords/ Retrieved:15-12-2018
28. Learning Precise Timing with LSTM Recurrent Networks Felix A. Gers, Nicol N. Schraudolph, Jrgen Schmidhuber; 3(Aug):115-143, 2002
29. Tyler Manning-Dahan (2017) PCA and Autoencoders
30. Chandra Mukkamala, Mahesh & Hein, Matthias. (2017). Variants of RMSProp and Adagrad with Logarithmic Regret Bounds.
31. van Wijk J. J. and van Selow E. R. Cluster and calendarbased visualization of time series data. In Proc. IEEE Symposium on Information Visualization, pages 4-9, Oct. 25-26, 1999.