UNIVERISTY OF PIRAEUS - DEPARTMENT OF INFORMATICS

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ – ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

MSC "Advanced Informatics and Computing Systems - Software Development and Artificial Intelligence"

ΠΜΣ «Προηγμένα Συστήματα Πληροφορικής – Ανάπτυξη Λογισμικού Και Τεχνητής Νοημοσύνης»

MSc Thesis

Μεταπτυχιακή Διατριβή

Thesis Title: Τίτλος Διατριβής:	**Deep Sequence Modelling of Data Envelopment Analysis - Based Performance Measurements** Ακολουθιακά Μοντέλα Βαθιάς Μάθησης για την Εκμάθηση Μέτρων Αποδοτικότητας Βασισμένα στην Περιβάλλουσα Ανάλυση Δεδομένων
Student's Name-Surname: Ονοματεπώνυμο Φοιτητή:	**Efthymios Makropoulos** Ευθύμιος Μακρόπουλος
Father's Surname: Πατρώνυμο:	**Ilias Makropoulos** Ηλίας Μακρόπουλος
Αριθμός Μητρώου:	ΜΠΣΠ20025
Supervisor: Επιβλέπων:	**Dionysios Sotiropoulos, Assistant Professor** Διονύσιος Σωτηρόπουλος, Επίκουρος Καθηγητής

November 2022 / Νοέμβριος 2022

3-Member Examination Committee

Τριμελής Εξεταστική Επιτροπή

Dionysios Sotiropoulos	**Evangelos Sakkopoulos**	**Georgios Tsichritzis**
Assistant Professor	**Associate Professor**	**Professor**
Διονύσιος Σωτηρόπουλος	Ευάγγελος Σακκόπουλος	Γεώργιος Τσιχριντζής
Επίκουρος Καθηγητής	Αναπληρωτής Καθηγητής	Καθηγητής

Acknowledgments

For this study, I would like to thank my family that has supported me throughout my school & university years and for providing me the physical and psychological means to achieve my goals.

Also, I would like to thank my supervisor, D. Sotiropoulos, with the help, guidance and mentoring of whom, I managed to deliver this project as accurately and as professionally as possible.

Abstract

Σε αυτήν την ενότητα παρουσιάζεται μια περίληψη της παρούσας εργασίας, καθώς και τα σημαντικά στοιχεία στα οποία βασίστηκε η διεκπεραίωσή της. Η ακόλουθη εργασία επικεντρώνεται στην εξερεύνηση και εξέταση διαφορετικών αλγορίθμων Τεχνητής Νοημοσύνης, οι οποίοι μπορούν να διαχειριστούν τον μετασχηματισμό ενός συνόλου δεδομένων από ένα πρόβλημα υπολογισμού efficiency scores στο πεδίο του Data Envelopment Analysis (DEA). Αρχικά παρουσιάζεται μια εισαγωγική έρευνα στο πεδίο της Τεχνητής Νοημοσύνης, συνεχίζοντας με την παρουσίαση του ερευνητικού πεδίου του Data Envelopment Analysis, ενώ ακολουθούν σχετικές έρευνες που έχουν γίνει και προσπάθησαν να συνδυάσουν το πεδίο έρευνας του DEA με αλγορίθμους Μηχανικής Μάθησης.

Τέλος, αφού παρουσιαστεί ο τρόπος σκέψης πίσω από τον μηχανισμό και ο αλγόριθμος με τον οποίο η παρούσα εργασία θα προσεγγίσει το πρόβλημα, γίνεται η καταγραφή της απόδοσης του μοντέλου που παρουσιάζεται, καθώς και μία σύντομη καταγραφή συμπερασμάτων από το αποτέλεσμα της προσομοίωσης εκτέλεσης αυτού του αλγορίθμου.

In this section of the study, a summary of the project and its key areas are presented. The following study is focusing on exploring and proposing different algorithms in the Artificial Intelligence scope that can manipulate and handle the data transformation from a dataset of a Data Envelopment Analysis efficiency scores' prediction problem, to conclude what are the key elements that make up this score and later to determine whether this kind of scores prediction is possible or not. The project starts with some introduction on the area of the Artificial Intelligence field of Computer Science, continues with a brief presentation of the DEA operations research technique and similar studies that have tried to combine the DEA research together with the Machine Learning sub-field from Artificial Intelligence.

Concluding, having presented the approach on the thinking behind the creation of the model to resolve this Machine Learning problem, the benchmarking of the performance of the model is presented, following by a brief description of the conclusions that were reached from the simulation of this algorithm execution.

Contents

Acknowledgments 3

Abstract 4

List Of Figures 7

List of Tables 9

Chapter 1 Introduction 10

1.1. Chronology of Machine Learning 10

1.2. Classification Problems and Machine Learning 10

1.3. Supervised & Unsupervised Machine Learning 11

1.4. Neural networks in Machine Learning 12

1.5. Data Envelopment Analysis (DEA) 13

Chapter 2 Relevant Studies 14

Chapter 3 Long Short-Term Memory (LSTM) Networks 16

Chapter 4 Sequence to Sequence Models 19

Chapter 5 Implementation Technologies of Neural Networks 21

Chapter 6 Application 24

6.1 Architectural Structure 24

6.2 Resources Used 24

6.3 Data Used 26

6.4 Data Types of the Objects Studied 27

6.5 Libraries Used 27

6.6 Source Code 28

6.6.1 Problems faced in first implementations and model definitions 28

6.6.2 Workarounds 30

6.6.3 Final implementation 32

6.7 Performance Summary per modification 35
6.8 Conclusions extracted 37
Chapter 7 Future Items 38
References 39

List Of Figures

1 Machine Learning as a subfield of Artificial Intelligence (Sindhu, 2020) 10
2 Two samples for an image classification problem. Purpose of the problem is to determine that the sample (a) belongs to a different class than the sample (b). (Koutroumbas & Theodoridis, 2008) 11
3 The basic stages involved in the design of a classification system. (Koutroumbas & Theodoridis, 2008) 11
4 Graph demonstrating the runtime in minutes of a 1- and 5-layer neural network using a CPU as the environment and the GPU as the environment. (Buber & Diri, 2018) 12
5 Architecture of the model to speed up the calculations of the DEA scores using RNNs from (Sotiropoulos & Koronakos, 2020) 14
6 Diagrams of the different neurons' units of RNN, LSTM and GRU (Jiaxuan & Shunyong, 2022) 16
7 Representation of LSTM cell, with the black boxes marking the interval for each timestep (Heaton, 2017) 17
8 LSTM cell with its gating mechanisms. (Beaufays, n.d.) 18
9 Average E(i,j) for seven emotion categories; for each emotion, Ek(i,j) is averaged over all frames from all speakers of that emotion; Where AC is acoustic frequency channels and MC is modulation frequency channels (Wu, Falk, & Chan, 2011) 19
10 Sequence to Sequence model consisting of sentence encoder and sentence decoder with attention. (Liang et al., 2020) 20
11 Chart explaining the relation between the metrics of loss/epoch given a learning rate 21
12 Diagram of a Neural Network cell structure (Tsagkaris, Katidiotis, & Demestichas, 2008) 22
13 Diagram indicating the workflow of the model in this study, during its training phase 24
14 Matlab's "about" screen from Matlab R2016a Version 25
15 Gurobi Logo from Gurobi.com 25
16 Pycharm's "about" screen from the Pycharm 2022.1.3 Version 26
17 A sample of the 10x4 array of the Double formatted numbers used in the Dataset 26
18 A sample of the 10x1 array of Double formmated numbers representing the CRS scores of each 10x4 array mentioned above 27
19 Training Accuracy & Loss over epochs for the model using a static learning rate, showing fluctuations on the performance indicating the underfitting of the model for both training and testing accuracy. 29
20 Learning rate decay graph over epochs using Adam Optimizer. 30
21 Training Accuracy & Loss over epochs for the first model, indicating underfitting for both training and testing accuracy. 30

22 Graph showing the performance metrics of the algorithm for each epoch using 1 tensor on each LSTM layer. 33
23 Graph showing the performance metrics of the algorithm for each epoch using 17 tensors on each LSTM layer. 33
24 Graph showing the performance metrics of the algorithm for each epoch using 65 tensors on each LSTM layer. 34

List of Tables

Table1: " Performance per number of LSTM cells used, using Adam Optimization modified learning rate"..... 34

Table 2: "Performance per number of LSTM Layers used, using static learning rate "..... 34

Table 3: "Performance per number of tensors in the LSTM layers used with 2 LSTM layers in encoder/decoder format, using static learning rate and 35,000 samples "..... 34

Table 4: "Performance per number of tensors in the LSTM layers used with 2 LSTM layers in encoder/decoder format, using static learning rate and 105,000 samples "..... 35

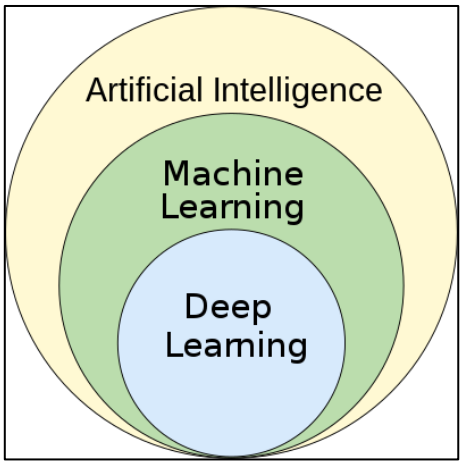
Chapter 1

Introduction

1.1. Chronology of Machine Learning

Machine Learning is a subfield of the Artificial Intelligence field in Computer Science and one of the most fast paced advanced fields of technology at the moment (Sindhu, 2020). Its applications and use cases are covering almost all fields of society where technology is used, starting from agriculture and education to healthcare and economics. (McQueen, Garner, Nevill-Manning, & Witten, 1995) (Varian, 2014)

The term of Machine Learning (ML) was firstly introduced in 1959 by IBM employee Arthur Samuel who had worked on Artificial Intelligence application for gaming. (Bowling, Fürnkranz, Graepel, & Musick, 2006) The synonym of it was "Self Teaching Computers and was used at the time but is not so popular anymore. (Lindsay, 1964).

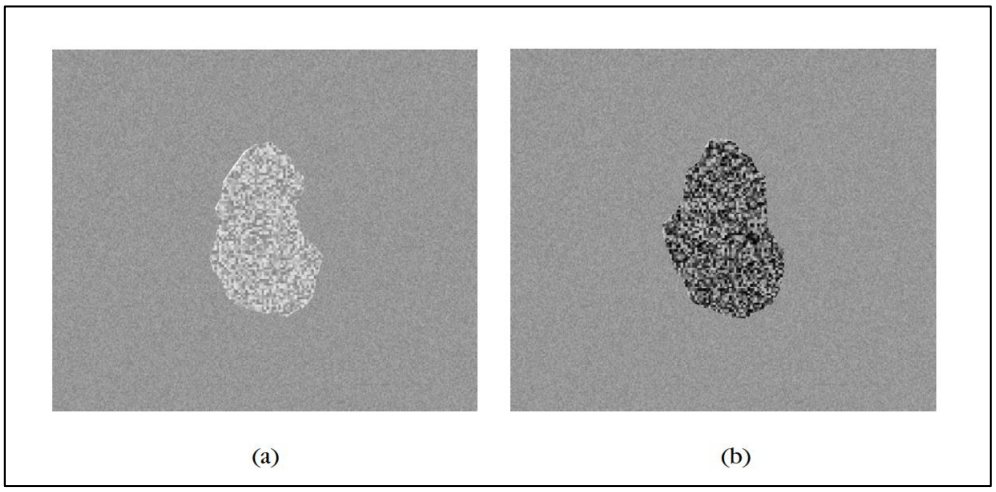


1 Machine Learning as a subfield of Artificial Intelligence (Sindhu, 2020)

The general purpose of machine learning is to create an algorithm that is using a set of data (dataset), in order to train and evaluate itself with it, for a specific problem of classification, data mining or trend prediction.

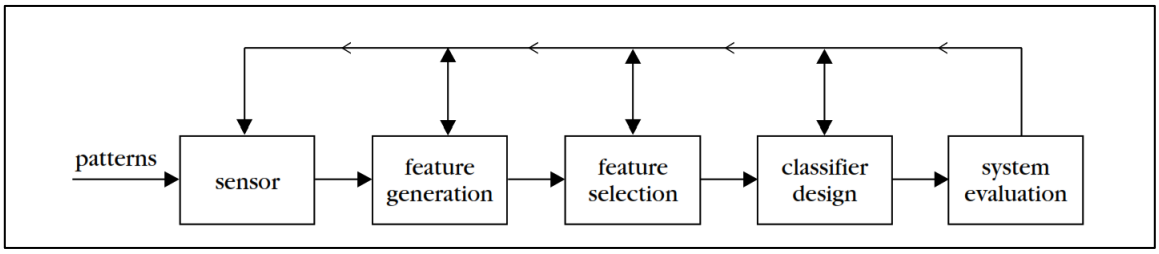
1.2. Classification Problems and Machine Learning

Problems which require a solution of a sample that -given a set of inputs- predicts an outcome belonging in 1 out of 2 or more categories are called classification problems. These kinds of problems are popular among the healthcare and bioinformatics science fields, and using Machine Learning algorithms to classify the dataset minimizes the time needed if the complete set of data is balanced and large enough to minimize the bias. (Nguyen, Bouzerdoum, & Phung, 2009) (Zhang, Wang, & Chen, 2014).



2 Two samples for an image classification problem. Purpose of the problem is to determine that the sample (a) belongs to a different class than the sample (b). (Koutroumbas & Theodoridis, 2008)

Methods called data-driven often use probabilistic statistics, machine learning or signal processing to firstly - using the dataset - collect the data and later select the most important features relevant to the classification problem to be solved. (Ban & Zhong, 2022). As it seems, the correct curation of the dataset is going to vastly impact the performance of the algorithm as well as the times of training.



3 The stages of a classification system representation. (Koutroumbas & Theodoridis, 2008)

1.3. Supervised & Unsupervised Machine Learning

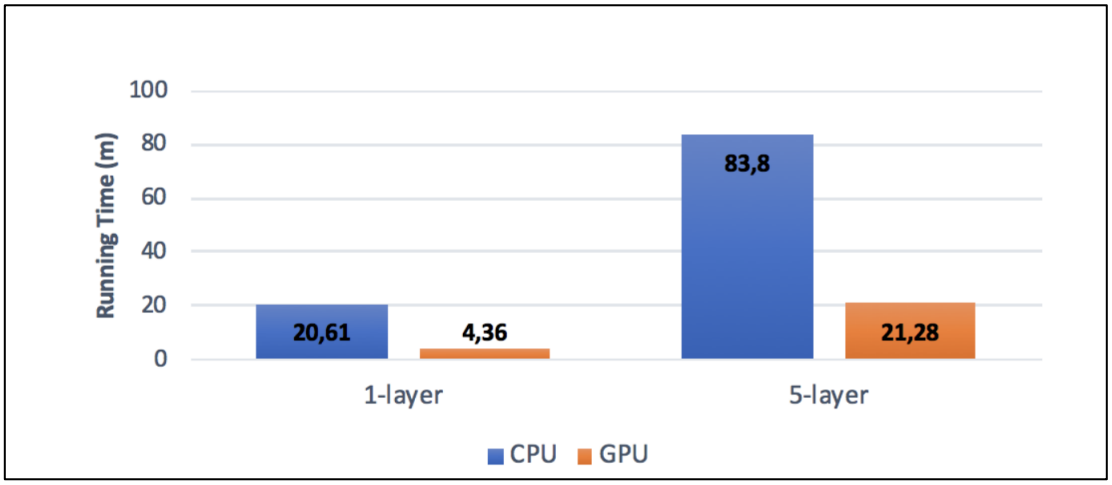
Classification and pattern recognition problems often fall under the scope of the Supervised Machine Learning branch. The machine learning algorithms used in these problems use the knowledge of pre-labeled datasets in order to be trained and then be evaluated. In other cases, Unsupervised Machine Learning models are used to discover similarities between a set of vectors of the samples and then group them together given a set of categories or classes. (Koutroumbas & Theodoridis, 2008)

In fact, combining the two methods described previously, a semi-supervised machine learning approach can be used by defining the theory – in a set of samples – if two points are close and linked by a path overlaying a high density area, then they are more probable to produce nearly located outputs and at the same time, if

the path that links them overlays a low-density region, their outputs are not required to be closely placed on the space. (Koutroumbas & Theodoridis, 2008)

1.4. Neural networks in Machine Learning

The advantage of the neural networks approach for machine learning is that there is a set of operations that can be performed simultaneously. With the computational cost that this had to induce in the model's requirements, the field was until recently marked with no big success. However, the situation was to change in the past few years with the advance of computational technology and the decrement of the cost of the Graphical Processing Units (GPUs). This has been proven to be critical in the performance of the model and the time needed for the operations needed to be performed by a neural network, since GPUs have more cores than the Central Processing Units (CPUs). (Buber & Diri, 2018) Thus increasing the performance of the machine learning models and making the neural networks able to successfully solve problems, which in the past were time-consuming or not-as-easy to be formulated. (Castillo, Cobo, Gutiérrez, & Pruneda, 1999) (Bishop, 1999)



4 Graph demonstrating the runtime in minutes of a 1- and 5-layer neural network using a CPU as the environment and the GPU as the environment. (Buber & Diri, 2018)

In this study, a set of libraries of different neural networks were used to compose the complete model that is going to be trained by the dataset and then be evaluated for the predictions to be done. The most famous implementation of the neural networks in deep learning is this for convolutional neural networks (CNNs). The CNNs have been leading the process of image recognition/interpretation, using the advantages of the convolutional deep layers of them in order to reduce the high dimensionality of the image's features without losing its information. (Donahue, et al., 2015)

In this study, the set of models used was focused on neural networks based on Long Short-Term Memory (LSTM) cells. LSTMs have been proven to work efficiently better in sequential problems used in a variety of use cases like Natural Language Processing (NLP) or real time speech recognition. (Krause, Murray, Renals, & Lu, 2017).

1.5. Data Envelopment Analysis (DEA)

Data envelopment analysis mechanisms were used to create the dataset of the application that is to be examined in the thesis. Data Envelopment Analysis (DEA) is a non-parametrical method which enables comparisons, where the units examined are using non easily discriminable resources ('inputs') to deliver outcomes ('outputs') on a large scale of characteristics, to yield a single measure of overall performance, so in general Data envelopment analysis constructs a composite KPI of the weighted sum of outputs to a weighted sum of inputs. (Thanassoulis & Conceição A. Silva, 2018).

There are two choices of orientation in DEA. There are models that are input oriented and output oriented. The aim of input orientation is to minimize the inputs at given output level and the aim of the output orientation is to maximize the output given at the input level. (Aziza, Mahadic, & Janorb, 2013).

In the DEA research operations, efficiency is the ratio of the output over the input that can be produced by, while comparing several entities/DMUs, depending on the complexity of the problem. However, when adding more inputs or outputs the efficiency computation becomes more complex. (Seiford, 1994) in their basic DEA model (the CCR) define the objective function to find DMU's efficiency defined as θ using the equation:

$$\max \quad \theta_j = \frac{\sum_{m=1}^{M} y_m^j u_m^j}{\sum_{n=1}^{N} x_n^j v_n^j},$$

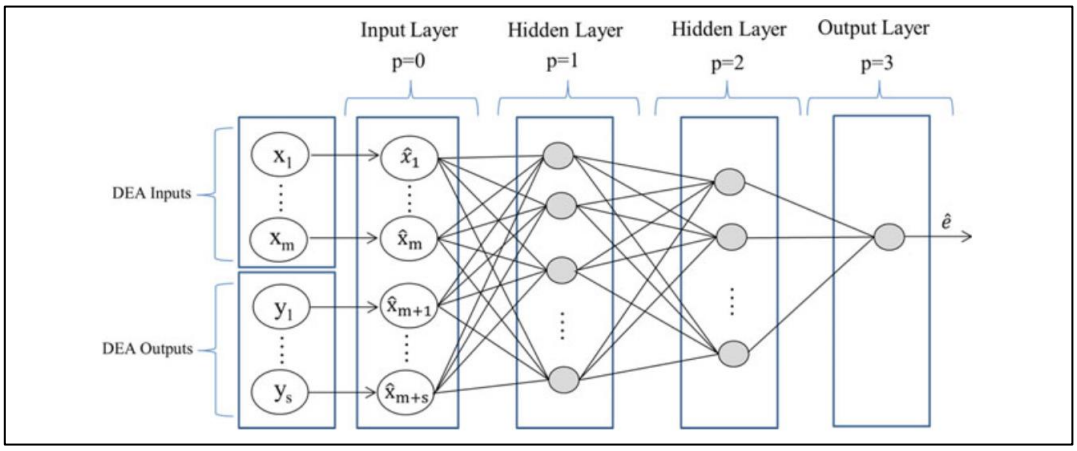
For the application of this study to generate the data for the training and the testing of the application, the Constant Returns to Scale (CRS) metrics of Input Oriented DEA models were used to generate matrices of random data units and link them to their CRS score.

Using then back propagation algorithms, the purpose is to check if an application, trained by a labeled dataset of vectors, can discover relationships between a -randomly - generated matrix of multiple vectors and the correct CRS score of each vector without knowing the CRS calculation formula.

Chapter 2

Relevant Studies

As this study is more focused on the DEA efficiency metrics of a dataset, several studies have approached these topics before with among the successful ones being this from (Sotiropoulos & Koronakos, 2020), where they successfully developed a new approach that could provide a mixture of DEA with Neural Networks to speed up the process of the calculation/evaluation efficiency scores in a DEA model. Later this year, their study (Sotiropoulos & Koronakos, 2020) also featured in G. Tzihritzis and J. Lain's book of Machine Learning Paradigms, where they further evaluated this study by presenting a feed forward neural network consisting of 1 input, 2 hidden and 1 output layer, making the approach more universal and abstract so that it can be applied in a larger area of science fields where similar solutions are required.



5 Architecture of the model to speed up the calculations of the DEA scores using RNNs from (Sotiropoulos & Koronakos, 2020)

Relevant studies on the field of Machine Learning models using LSTM have been done by (Shi, et al., 2015) on the topics of the weather forecast combining artificial intelligence technologies, by proposing new LSTM layers for the training of their algorithm. Interesting studies on this topic and related to the hardware efficiency of Solid Stade Drives' (SSDs) Quality of Service improvements can be found in the projects from (Chakraborttii, Sinha, & Litz, 2018), where using LSTM recurrent neural networks, the algorithm was successfully (~82% accuracy) trained to learn spatial patterns from the input-output traces of the block level data transfer. Later in 2019, (Borovkova & Tsiamas, 2019) were able to successfully propose an alternate version of the LSTM NNs for the stock predictions purpose in an online platform to examine the importance of the characteristics affecting the stock market.

Sequence-to-Sequence models are still mostly popular for the text prediction or text composition, or text classification problems in general, that require real-time human to machine language data points translation. In 2015 (Ray, Rajeswar, & Chaudhury, 2015) proposed a bi-directional alternation of the LSTM architectures, that could recognize Oriya (also known as Odia) text characters (characters are not of the latin alphabret), that are used as a preliminary India-Aryan language alphabet. In this study, modifying the LSTM architecture and applying CTC (Connectionist Temporal Classification) for the training process, the algorithm

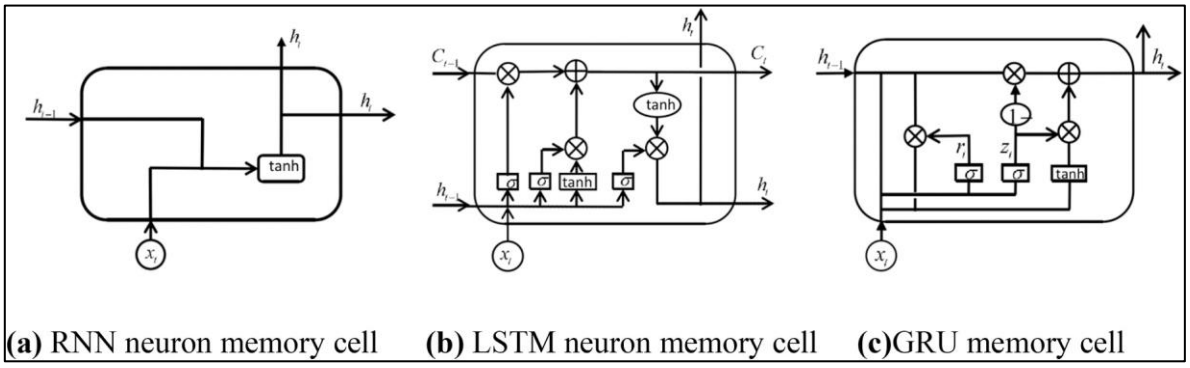
could achieve accuracy in the likes of less than 5% for character recognition and less than 13% for word recognition on printed Odia text.

Further applying the LSTM RNNs in 2015, (Messina & Louradour, 2015) could successfully present a multi-dimensional LSTM network that could recognize handwritten Chinese text (ping-ying ideograms) without forcefully applying segmentation of the characters.

Chapter 3

Long Short-Term Memory (LSTM) Networks

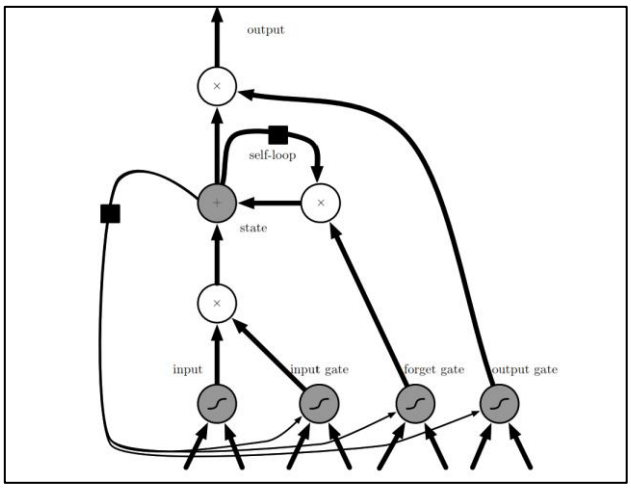
Long Short-Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997) and Gated Recurrent Units (GRUs) (Chung, et al., 2014) are new variants of RNNs, have solved long-term dependency problems, and have attracted great attention. LSTM's variant of the Recurrent Neural Networks were proposed for problems that needed the information of the importance of a term met in a long sequence of data. The advantage of an LSTM model is that it uses the additional cell state, which is a horizontal sequence line on the top of the LSTM unit. This cell state is controlled by special purpose gates for forget, insert, or update operations. (Yıldırım & Asgari-Chenaghlu, 2018)



6 Diagrams of the different neurons' units of RNN, LSTM and GRU (Jiaxuan & Shunyong, 2022)

Long shot-term memory network or LSTM network is an artificial recursive neural network using feedback connections to train itself using sequential short-term memories. (Monner & Reggia, 2012). Having a variety of use, LSTM networks seem to perform well in text prediction problems inside a sequence. (Islam, Mousumi, Abujar, & Hossain, 2019), making it a good candidate to be chosen for the problem of predicting a sequence of numbers generated by the DEA efficiency metrics' calculation algorithm described earlier.

Other use cases for LSTM network include unconstrained handwriting recognition (Graves et al., 2009), speech recognition (Graves et al., 2013; Graves and Jaitly, 2014), handwriting generation (Graves, 2013), machine translation (Sutskever et al., 2014), image captioning (Kiros et al., 2014b; Vinyals et al., 2014b; Xu et al., 2015), and parsing (Vinyals et al., 2014a). The LSTM block diagram is illustrated in figure 10.16. The corresponding forward propagation equations are given below, for a shallow recurrent network architecture. Earlier (Graves et al.,2013; Pascanu et al., 2014a) managed to deliver architectures that have more hidden layers or are trained in the deeper stage of the output of the neuron. Meaning that instead of a unit that simply applies an element-wise nonlinearity to the affine transformation of inputs and recurrent units, LSTM based recurrent networks have "LSTM cells" that are iterating over themselves in order to achieve a self-training over a short-memory of a sequence they are fed. Then each cell forwards the output to the rest of the RNN mechanism. (Heaton, 2017)



7 Representation of LSTM cell, with the black boxes marking the interval for each timestep (Heaton, 2017)

LSTMs at each time step can read or write the cell using explicit gating algorithm. The precise formula of the update is defined below: (Karpathy, Johnson, & Li, 2015)

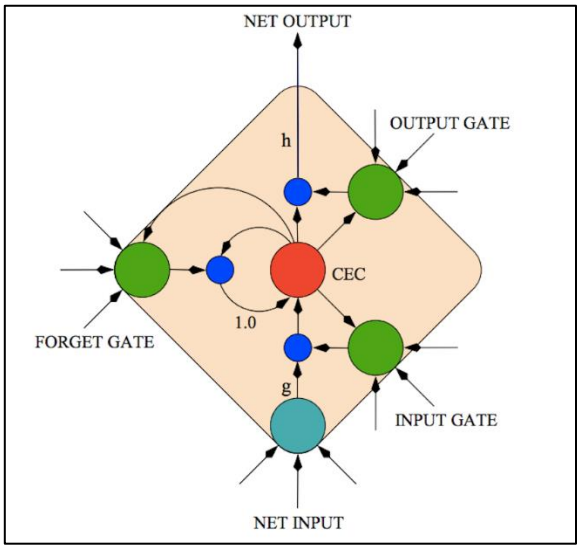
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} sigm \\ sigm \\ sigm \\ sigm \end{pmatrix} \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^{l} \end{pmatrix} \qquad \text{where:} \qquad c_t^l = f \times c_{t-1}^l + i\, g, \quad h_t^l = o \times \tanh(c_t^l)$$

Implementations in different programming languages can be found for the LSTM networks, however since this study focuses on the programming language of Python 3.6 and Python 3.10, the package of the open source keras library (keras.io) is going to be used for the LSTM neurons of the network. A coding sample that implements the above described way of working for the LSTM cells can be found below:

```
def standard_lstm(inputs, init_h, init_c, kernel, recurrent_kernel, bias,
                  mask, time_major, go_backwards, sequence_lengths,
                  zero_output_for_mask):

  input_shape = backend.int_shape(inputs)
  timesteps = input_shape[0] if time_major else input_shape[1]

  def step(cell_inputs, cell_states):
    """Step function that will be used by Keras RNN backend."""
    h_tm1 = cell_states[0]  # previous memory state
    c_tm1 = cell_states[1]  # previous carry state

    z = backend.dot(cell_inputs, kernel)
    z += backend.dot(h_tm1, recurrent_kernel)
    z = backend.bias_add(z, bias)

    z0, z1, z2, z3 = tf.split(z, 4, axis=1)

    i = tf.sigmoid(z0)
    f = tf.sigmoid(z1)
    c = f * c_tm1 + i * tf.tanh(z2)
    o = tf.sigmoid(z3)
```

```
  h = o * tf.tanh(c)
  return h, [h, c]

last_output, outputs, new_states = backend.rnn(
    step,
    inputs, [init_h, init_c],
    constants=None,
    unroll=False,
    time_major=time_major,
    mask=mask,
    go_backwards=go_backwards,
    input_length=(sequence_lengths
                  if sequence_lengths is not None else timesteps),
    zero_output_for_mask=zero_output_for_mask)
return (last_output, outputs, new_states[0], new_states[1],
        _runtime(_RUNTIME_CPU))
```

LSTM recurrent neural networks are now used as the optimal recurrent network for sequential recognition problems in Machine Learning (e.g. Google Voice Transcription Model System). This is because LSTM networks have additional recurrent connections and memory cells that make them able to keep track of the importance of each data input they have processed. (Beaufays, n.d.).



8 LSTM cell with its gating mechanisms. (Beaufays, n.d.)

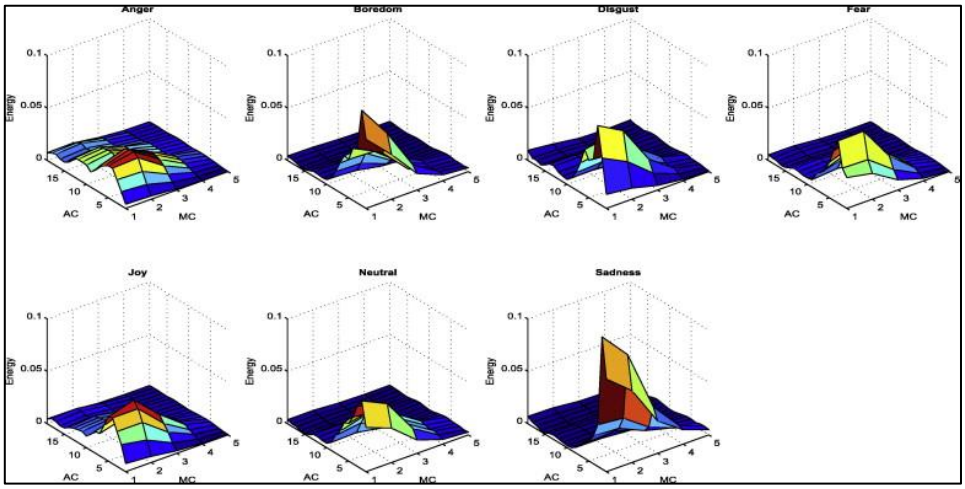
Chapter 4

Sequence to Sequence Models

Sequence-to-Sequence models (also mentioned as Seq2Seq) have been proven to perform well in tasks of automatic speech recognition (ASR), speech synthesis, chatbots or text prediction problems due to their ability to focus on specific elements' importance of a given input vector, or in speech context, focusing on specific words' importance of a given sentence. For example, there are studies that have been focusing on improving the performance of such mechanisms of speech recognition using sequence-to-sequence models and editing their training procedures in order to reach peak performance with really high accuracy given a set of samples. (Chung-Cheng etal, 2018)

Network based text-to-speech models can be implemented by using recurrent Seq2Seq complex architectures that can synthesize sounding speech directly by labelling text characters using attention techniques of the encoder-decoder approach. (Wang, 2019)

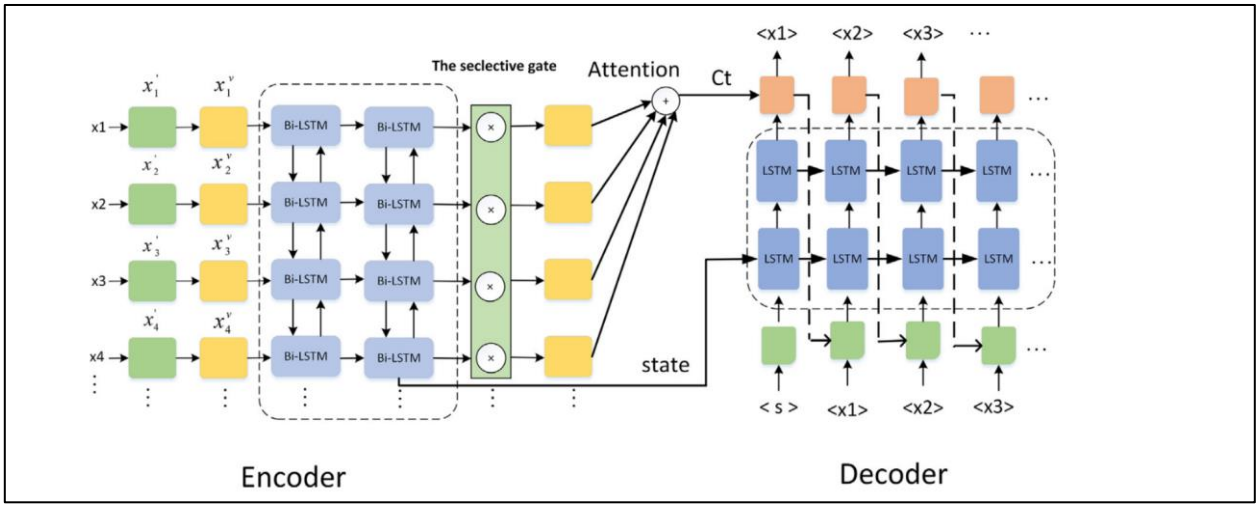
Other studies that use Sequence to Sequence models are focusing on the details of the tone of the voice sample. For example, study from (Wu, Falk, & Chan, 2011) is using models that are focusing on extracting emotion characteristics out of automatic speech recognition samples using characteristics extraction techniques like modulation spectral features.



9 Average E(i,j) for seven emotion categories; for each emotion, Ek(i,j) is averaged over all frames from all speakers of that emotion; Where AC is acoustic frequency channels and MC is modulation frequency channels (Wu, Falk, & Chan, 2011)

In the present study, using abstract programming theory, the sequence-to-sequence model architecture can be used, to manipulate each given dataset sample of vectors of 4 integers inside an array of 10 vectors, like a word inside a sentence and using semi-supervised machine learning with labeling of CRS scores, train the model to accurate predict a CRS score (label) given a randomized array of vectors (sentence).

Further modifying the Sequence-to-Sequence models using LSTM networks, alternations like the attention technique, have been found to be improving the performance and the accuracy of the models. In encoder-decoder models, concatenate attention mechanism can be used to compute the context vector of each state. In each next state the previous decoder state with each encoder hidden state have to be aligned to get the importance score and then the attention scores can be calculated through the normalization operation of the energy weight matrix, so that the context vector c_t output of the attention mechanism can be calculated. (Liang et al., 2020)



10 Sequence to Sequence model consisting of sentence encoder and sentence decoder with attention. (Liang et al., 2020)

The way of the sequence-to-sequence implementation methods for this project - using Python programming language and its libraries - can be found on Chapters 5 and 6 later in this study.

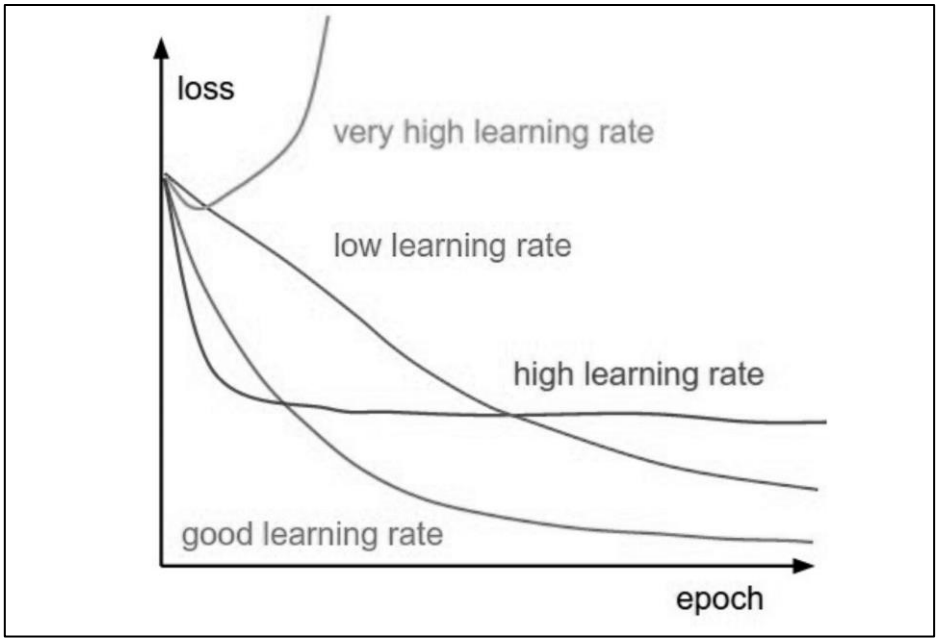
Chapter 5

Implementation Technologies of Neural Networks

Any neural network has a specific set of hyperparameters that are to be manually adjusted by the engineer every time something needs to be changed and they are not updated by the model. These hyperparameters are the following:

- **Number of hidden layers**: The hidden layers of a neural network are located between the input and the output of the network cell. The activation function, definition follows later in this chapter, calculates the output as a transformation of the inputs of the cell using the weighted input as a parameter. The idea is that any NN should be as simple as possible (for performance purposes it needs to be fast and also generalized for scalability), but at the same time it needs to adjust properly to the problem and classify the given input data. (Shafi, Ahmad, Shah, & Kashif, 2006).

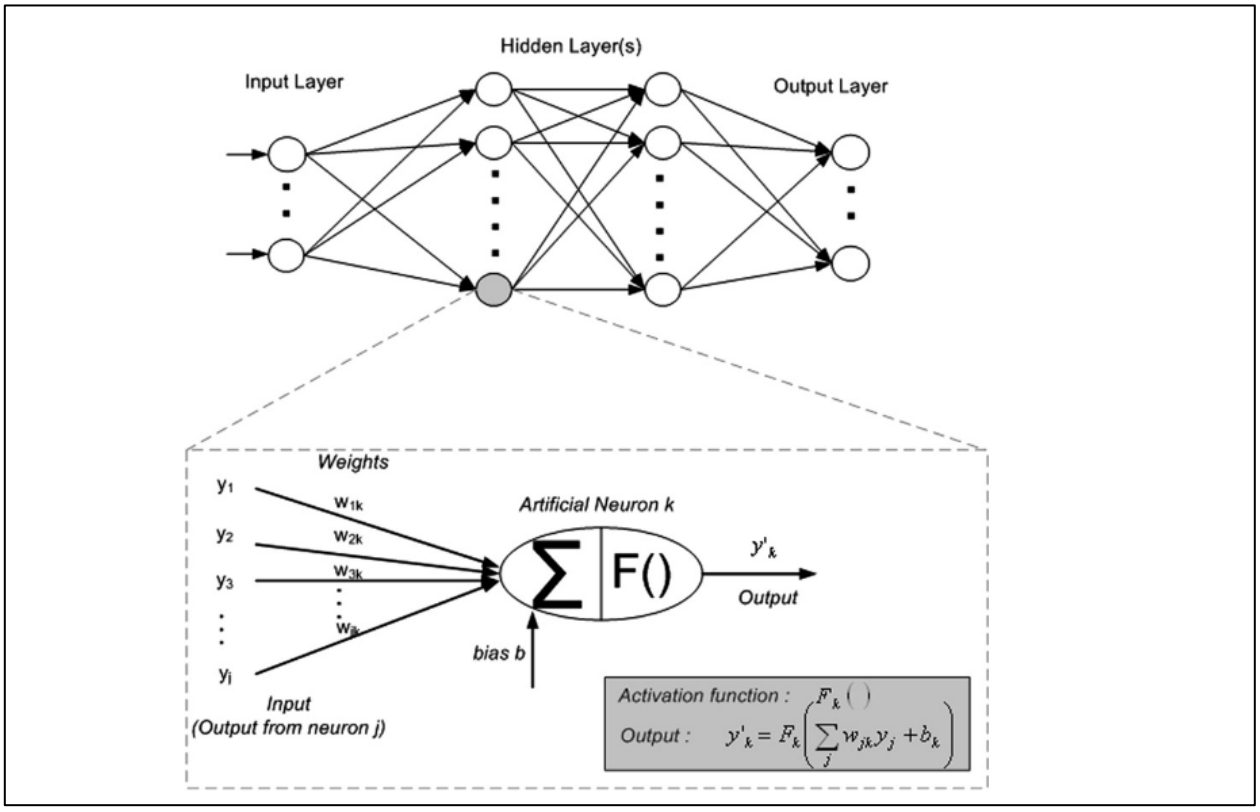
- **Learning rate**: The learning rate parameter is the pace that the NN is going to use in order to update the weights applied in the inputs in the process of the hidden layers calculations. This parameter of the NN model is mainly empirical and needs to be modified accordingly for any given problem to b make the model converge towards the learning with the optimal step. Two risks of applying the learning rate too high or too low is the convergence rate of the model towards the accurate results. With very low learning rate the model will converge slowly, implying the risk of the Vanishing Gradient problem. With very high learning rate the possibility of missing the minimum error increases, producing the Exploding Gradient problem (Philipp, Song, & Carbonell, 2017).



11 Chart explaining the relation between the metrics of loss/epoch given a learning rate

- **Momentum**: In back propagation algorithms, the technique of momentum can be introduced to improve the speed of the convergence. Momentum is the technique of keeping memory of the previous directions of the parameters of the learning rate, in order to calculate and recalibrate a new learning rate slightly modified.

- **Activation function**: During the training phase, for the model to transform the weighted inputs and produce the output of the cell, the activation function is introduced. In most of the cases, the model needs to have the function produce an output in the range of (0,1) in order to do the classification/labeling of the sample. For this reason, sigmoid functions like “Softmax”, “ReLU”, “RBF”, “tanh” and others are very popular as activation functions that given any input in (-∞, +∞) produce outputs in the range of (0,1) which is then considered the probability of this set to belong to the class it was calculated upon.



12 Diagram of a Neural Network cell structure (Tsagkaris, Katidiotis, & Demestichas, 2008)

- **Batch Size:** Models that are supposed to be trained with a big amount of data samples, have been proven to be inefficient when they are fed with the whole set of the dataset at one go. So the batch (or mini-batch size) is introduced as the size of the inputs of the training neurons of the model. The predominant option is to choose 32 as the batch size, but it is relevant to each problem and not a canon.

- **Epochs:** The epochs characteristic of the Neural Network is defined as the number of times the algorithm is going to train the model on the whole dataset. Same as the learning rate, this is a parameter that needs to be empirically experimented upon to find the optimal option for each problem. In this study the model has its epochs halt each time the losses between the current epoch and the previous one is reaching close to a marginal error.

- **Dropout:** The technique of Dropout is introduced during the training phase in order to remove some nodes that might be not contributing as others to the learning of the model. The implementation of this technique is to define a probability by which the model is either keeping or removing a node during each epoch.

For the model to be created with the characteristics mentioned above, the packages of Keras from the Tensorflow library were used, so that their neural network templates can be implemented in to the sequence-to-sequence model studied in this project.

On the following chapters, the way that these technologies will be implemented is going to be presented.

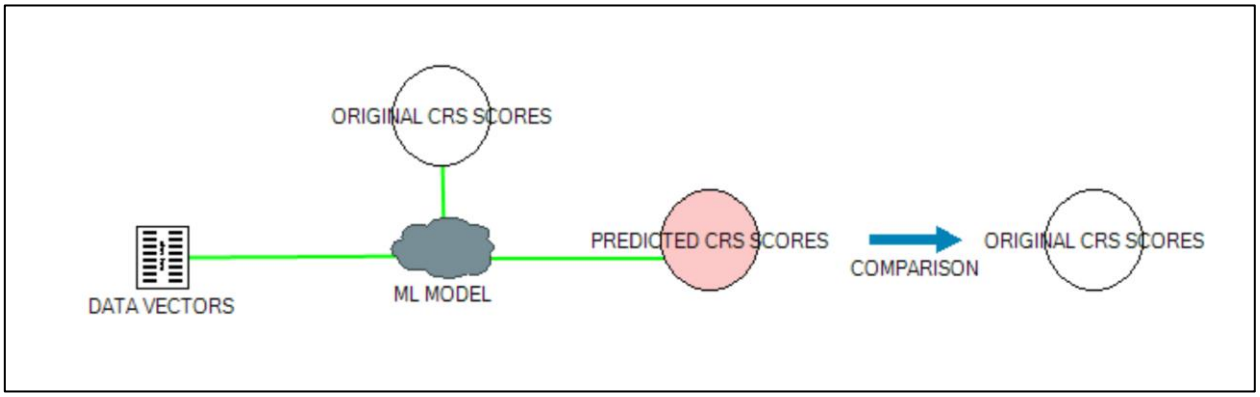
Chapter 6

Application

6.1 Architectural Structure

As described before, the dataset of this study is created using the DEA efficiency metrics scores of CRS for a given array of 10x4 elements.

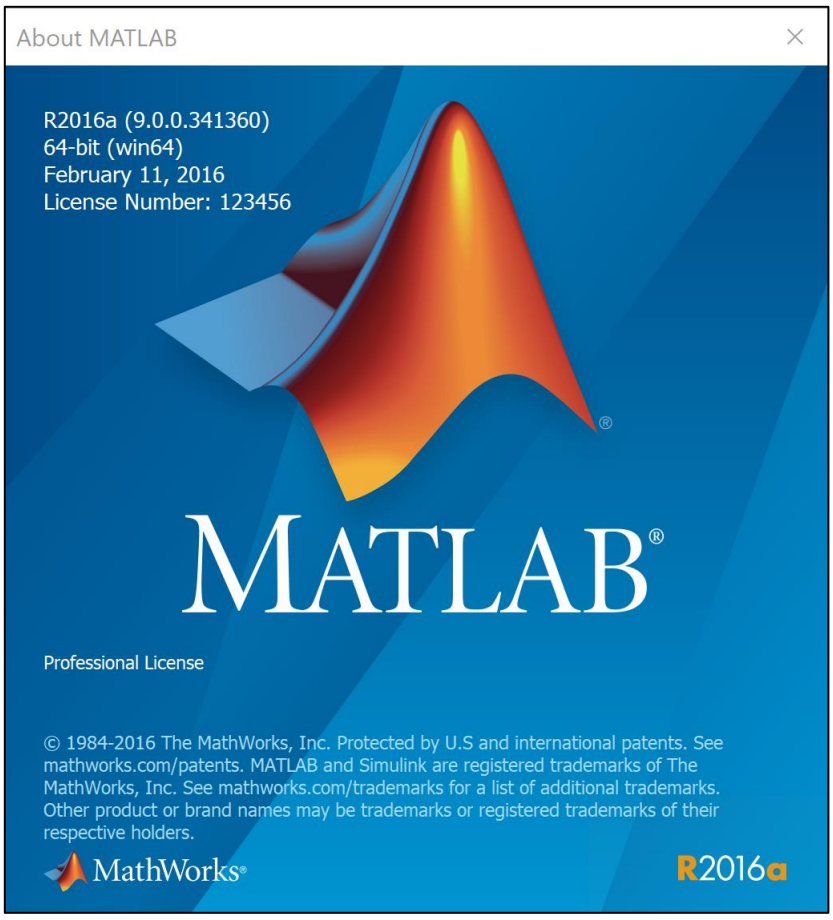
Given this dataset, the purpose is to pass it in large batches of data to the model in order to identify its ability of re-creating the CRS scores trying to converge them closer to the original CRS scores.



13 Diagram indicating the workflow of the model in this study, during its training phase

6.2 Resources Used

For the dataset creation, MATLAB v.: R2016 was used in to generate the matrices of the data and the CRS scores.



14 Matlab's "about" screen from Matlab R2016a Version

For the DEA metrics calculations, Gurobi Optimizer from Gurobi Optimization was used in order to generate the CRS and DRS scores of the data matrices.

15 Gurobi Logo from Gurobi.com

For the Python implementation of the model as well as the debugging of the code and the testing of the application, PyCharm IDE from Jetbrains was used and the source code was built in the programming language Python 3.9. (Python.org, 2020)

16 Pycharm's "about" screen from the Pycharm 2022.1.3 Version

6.3 Data Used

The data were generated using the Gurobi Optimization library for MATLAB to calculate their DEA efficiency metrics like the CRS/VRS scores of given arrays. Then these were stored in MATLAB .mat files that were later parsed by the model using the help of the Python open-source libraries mentioned later on this chapter.

	1	2	3	4
1	4.9487	3.4842	7.7614	8.5665
2	4.4340	7.1173	3.2959	3.2885
3	7.8897	6.8959	5.5536	8.3286
4	8.1568	2.4635	7.2917	3.1917
5	2.6819	2.0710	9.0181	9.3634
6	5.4079	5.4853	9.6336	4.1499
7	5.0103	9.6377	5.9249	2.7694
8	6.8168	4.0635	2.2476	3.2598
9	7.3843	6.2674	2.3436	6.5444
10	7.7922	3.0143	3.3176	5.2596

17 A sample of the 10x4 array of the Double formatted numbers used in the Dataset

	1
1	0.5438
2	0.2210
3	0.3024
4	0.6797
5	1
6	0.5298
7	0.3517
8	0.1774
9	0.2538
10	0.3859

18 A sample of the 10x1 array of Double formmated numbers representing the CRS scores of each 10x4 array mentioned above

For the model to be able to be trained properly, certain transformations on the arrays were made, to prepare the dataset to be fed to the Neural Network model created. For better understanding the transformations made and the reason behind them, a look in the code shall be taken.

6.4 Data Types of the Objects Studied

For the objects of the model to – properly - be represented in the python language, the following objects were used:

- Lists: Ordered collection/set of objects that are not required to belong on the same class
- Arrays: Containers of objects that can store multiple items in a 1-d, 2-d, 3-d space. Maximum dimensionality of the arrays using NumPy library is 32 dimensions.
- Dictionaries: A collection of objects stored in a "key: value" format.
- Plots: Images containing graphs for a given set of datapoints and parameters.

Using the below mentioned libraries, the mat files were parsed to firstly split the files in Data and CRS dictionaries and then acquire their values into matrices/arrays - that were later converted to lists for their easier modifications/manipulations.

The plots were then used to graphically represent the performance of each model's iteration.

6.5 Libraries Used

Using the import statement from Python, this project implements open source and custom-made using the following packages and their libraries.

```
import sys
```

Sys library was used for debugging purposes – to halt the code execution

```
import time
```

Time library was used for debugging and performance calculation purposes

```
import os
```

OS package was used to allow the program to parse through the dataset files

```
import scipy.io
```

Scipy.io package was used to load the matrices of the dataset

```
import numpy as np
```

Numpy library was used to manipulate the matrices of the dataset

```
import tensorflow as tf
import keras as ke
```

Keras and **Tensorflow** packages were used to implement the neural network architectures of LSTM, CNN in to the model

```
from sklearn.model_selection import train_test_split
```

Sklearn package was used to split the dataset to training and testing according to the 90-10 rule

6.6 Source Code

Source code and its dependencies, as well as the dataset used, can all be found in the root project directory - submitted through this study - and in GitHub: https://github.com/eft28/s2s. The source code is written in Python programming language using the OOP (Object Oriented Programming) approach.

6.6.1 Problems faced in first implementations and model definitions

Starting with simpler models and using fewer LSTM cells, the model was trained to get rewarded for the minimum categorical cross-entropy, which is a mixture of the softmax activation function and the cross-entropy loss function, in order to allow the model to generate a vector of probabilities of classification for each class that a sample might belong to. So, the mathematic formula that handles the activation function for these LSTM cells in the training can be written like this:

$$CE = -\sum_{i}^{C} t_i log(s_i)$$

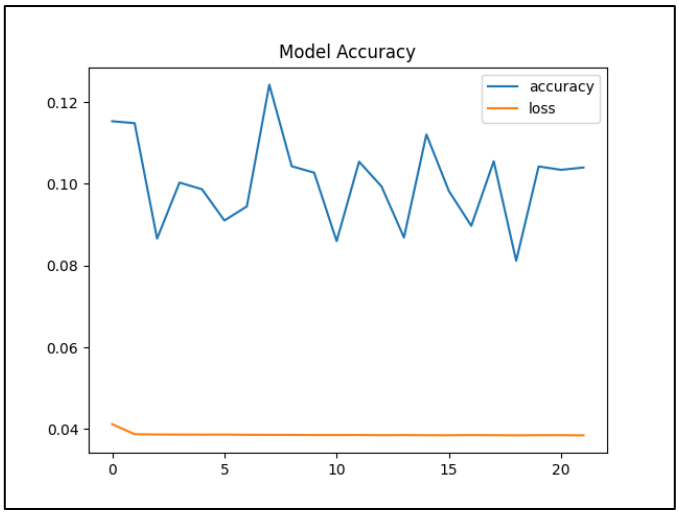
CE is the crossentropy value, for a C class, where t_i is the actual value getting logarithmic multiplication against the predicted value of s_i.

However, by the nature of its problem, this way of that the activation functions worked, led the model to face the problem of underfitting, as the accuracy of its training soon stopped increasing above 30% and could not generalize in new data, as shown in the performance graphs of the figures following.

The code representation in Python for the model definition consisted of 1 input layer, 1 simple LSTM layer and 1 output layer:

```
model = Sequential()
model.add(Input(shape=inputs_shape))
model.add(LSTM(128, return_sequences=False))
model.add(Dense(units=10, input_shape=(4, 10), activation="sigmoid"))
```

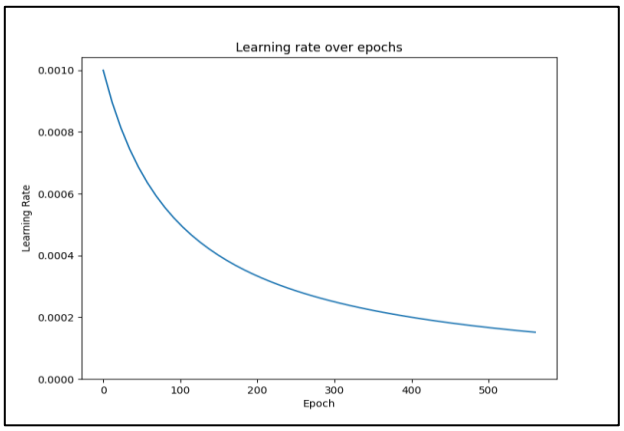
Using static learning rate, showed that the model could not really be trained efficiently to sort the arrays to each



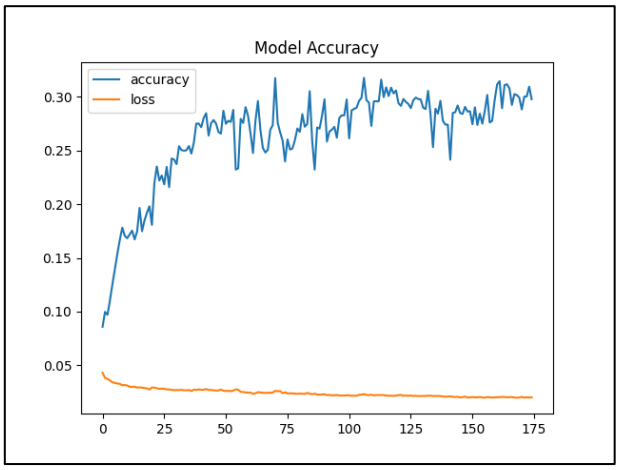
19 Training Accuracy & Loss over epochs for the model using a static learning rate, showing fluctuations on the performance indicating the underfitting of the model for both training and testing accuracy.

In order to further improve the model towards its training performance, slightly modifying its learning rate using the Adam optimization algorithm saw that the results were more optimistic. Adam optimization algorithm is the process of slightly decreasing the learning rate during each epoch in order to allow the model to converge to the optimal point in less time, compared to using a static learning rate co-efficient. (Bae, Ryu, & Shin, 2019). The way in which the Adam optimizer is written in python code in this project, follows in the snippet below:

```
lr_schedule = optimizers.schedules.InverseTimeDecay(
    0.001,
    decay_steps=steps_per_epoch * 100,
    decay_rate=1,
    staircase=False
```



20 Learning rate decay graph over epochs using Adam Optimizer.



21 Training Accuracy & Loss over epochs for the first model, indicating underfitting for both training and testing accuracy.

6.6.2 Workarounds

As described above, the loss function used for the training can be chosen differently, to examine the scenario of the model to get rewarded not over the correct guessing of a specific set of probabilities for the classes (where classes are represented as a CRS score for each vector), but to be trained for a binary classification problem between each of the classes that the sample can belong to. Meaning that for each class available for prediction, the model is using a binary set of 2 classes that determine if the sample can belong or not belong to this class using a probabilities vector. Using this activation function, the problem is getting the subtask of identifying the probability for each of the given classes, converting it partially to a binary classification problem.

Further escalating to the mathematic representation of this binary classification modification of the cross-entropy activation function, the following formula of cross entropy is used:

$$CE = -\sum_{i=1}^{C'=2} t_i log(f(s_i)) = -t_1 log(f(s_1)) - (1 - t_1) log(1 - f(s_1))$$

Aiming to improve the model's efficiency and performance on the dataset, the decision to add more layers of LSTM was taken. In that case, with more LSTM layers, the model is allowed more inputs in each iteration to "decipher" the complex data-crs correlation.

Using this modification of CE for the activation functions the following 3 models were created. The description of the models and their summary representation from Keras Library is following:

- Model with 1 layer of LSTM between 1 input and 1 output Layer without encoder/decoder stack.

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 128)	71168
dense (Dense)	(None, 10)	1290

- Model with 3 layers of LSTM with a Batch Normalization Layer in between and a Dropout layer following.

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 128)	48
Dropout (Dropout)	(None, 1)	0
lstm_2 (LSTM)	(None, 1)	24
BN (BatchNormalization)	(None, 10)	0
lstm_3 (LSTM)	(None, 1)	12

dense_1 (Dense)	(None, 10)	20

6.6.3 Final implementation

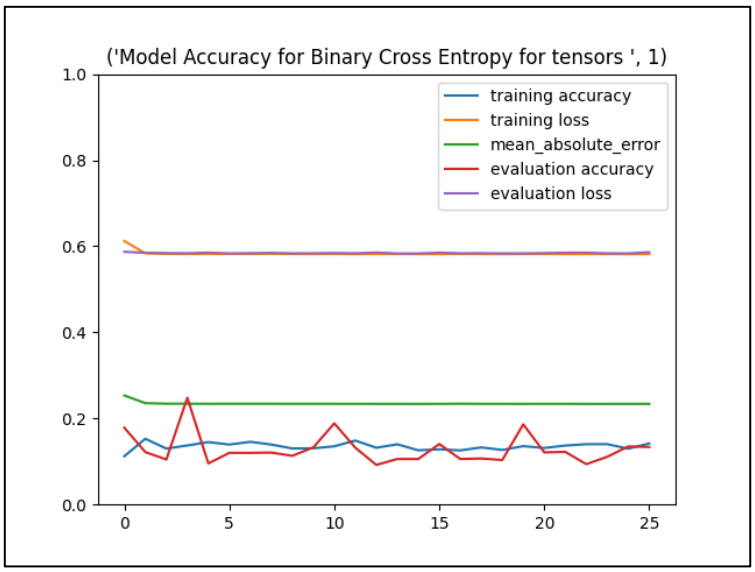
The final implementation of the model is consisted of the encoder decoder mechanism to implement the sequence-to-sequence model representation without using the attention technique. In order to achieve this architecture, the lstm layers are stacked in continuation of one another, with a repeat vector layer in between to make the outputs of the first cell connectable to the inputs of the next in the reverse way.

Model with an encoder/decoder mechanism and an output layer.

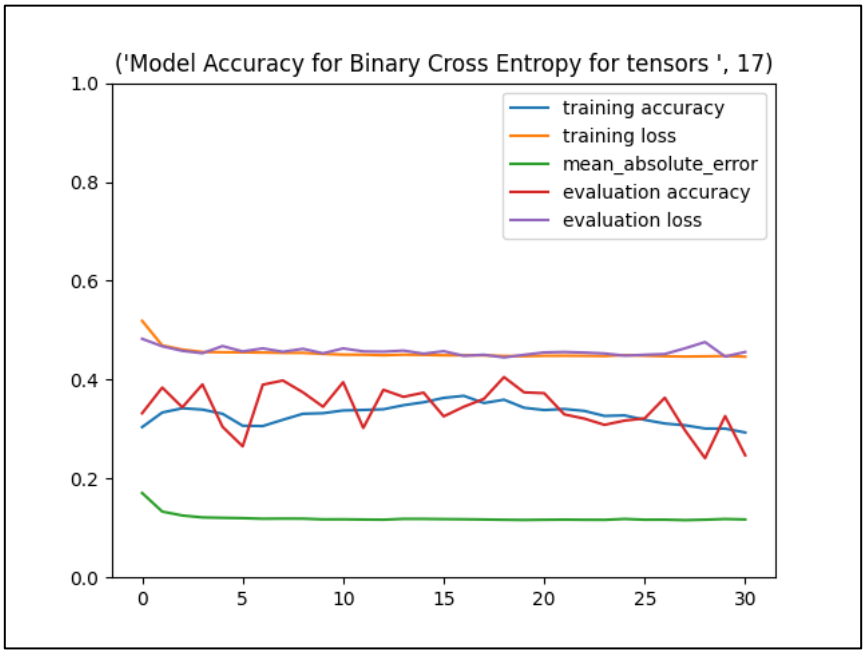
Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 1)	48
r v (RepeatVector)	(None, 10, 1)	0
lstm_2 (LSTM)	(None, 1)	12
dense_1 (Dense)	(None, 10)	20

Using this solution with the modifications mentioned before, the model saw greater success on predicting the CRS scores during training and during the evaluation process as well. In order to also identify the actual loss of the prediction, the algorithm calculated the mean absolute error, which calculates the absolute difference between the predicted CRS and the actual CRS score of each array, during each iteration.

On the process to identify other important characteristics of this dataset, the code was modified to loop over a number of tensors that can be used on the first layer of LSTM feeding. So, the algorithm given a specific number of loops, went to add and examine each model with a different number of tensors each time. Little progress was identified so there was not enough proof that this modification changed the performance to be improved.



22 Graph showing the performance metrics of the algorithm for each epoch using 1 tensor on each LSTM layer.



23 Graph showing the performance metrics of the algorithm for each epoch using 17 tensors on each LSTM layer.

24 Graph showing the performance metrics of the algorithm for each epoch using 65 tensors on each LSTM layer.

6.7 Performance Summary per modification

In this section, the tables that summarize the performance per each model characteristics are presented in order to make more clear the comparisons between them, as well as to emphasize the progress of this study during the different problems that were faced.

Samples	LSTM layers	Encoder-Decoder ?	Training Accuracy	Evaluation Accuracy	Mean Absolute Error (MAE)
10,000	1	No	0.085423	0.005434	0.42123
10,000	2	No	0.115424	0.004521	0.35976
35,000	1	No	0.156935	0.014502	0.48125
35,000	2	No	0.183552	0.059421	0.32217
105,000	1	No	0.162271	0.024124	0.36423
105,000	2	No	0.183532	0.062311	0.31721

Table 1: Performance per number of LSTM cells used, using Adam Optimization modified learning rate

Samples	LSTM layers	Encoder-Decoder ?	Training Accuracy	Evaluation Accuracy	Mean Absolute Error (MAE)
10,000	1	No	0.104241	0.054269	0.44168
10,000	2	No	0.185716	0.159981	0.38966
35,000	1	No	0.219521	0.178828	0.31001
35,000	2	No	0.290185	0.242161	0.29888
105,000	1	No	0.28991	0.212511	0.31664
105,000	2	No	0.291241	0.288772	0.29888

Table 2: Performance per number of LSTM Layers used, using static learning rate

Samples	LSTM input tensors	Encoder-Decoder ?	Training Accuracy	Evaluation Accuracy	Mean Absolute Error (MAE)
35,000	1	Yes	0.291241	0.28001	0.341221
35,000	17	Yes	0.430401	0.39001	0.290056
35,000	33	Yes	0.420988	0.41068	0.260951
35,000	49	Yes	0.416691	0.41241	0.240001
35,000	65	Yes	0.395125	0.38511	0.240001
35,000	81	Yes	0.412151	0.40005	0.213444

Table 3: Performance per number of tensors in the LSTM layers used with 2 LSTM layers in encoder/decoder format, using static learning rate and 35,000 samples

Samples	LSTM layers	Encoder-Decoder ?	Training Accuracy	Evaluation Accuracy	Mean Absolute Error (MAE)
105,000	1	Yes	0.340124	0.340001	0.249111
105,000	17	Yes	0.395112	0.380212	0.265512
105,000	33	Yes	0.458881	0.4492142	0.215621
105,000	49	Yes	0.458221	0.449981	0.191521
105,000	65	Yes	0.441242	0.415125	0.221124
105,000	81	Yes	0.468124	0.442211	0.258812

Table 4: Performance per number of tensors in the LSTM layers used with 2 LSTM layers in encoder/decoder format, using static learning rate and 105,000 samples

6.8 Conclusions extracted

In conclusion, the experiments that were produced during this study showed that the problem described is indeed a sequence-to-sequence problem and there is evidence aiming the direction of it to be handled by algorithms that are focusing on these specific sets of problems. The approach was deterministic, in terms of the means that were used and mainly empirical in order to identify the optimal kit of tools which could be used to handle such data manipulations.

Moving away from the 1-d LSTM layers Neural Networks and closer to more complex architectures like that of the encoder-decoder mechanism showed that the accuracy of the model actually increased, and the explosive gradient problem was resolved. This led the project towards a solution more closely related with the sequence-to-sequence machine learning models that handle similar problems, using this approach.

In addition, adding more samples to the training dataset as well as permutating the dataset to make it more immune to the noise that is being affected by, further increased the speed to which the algorithm converged to the optimal solution it could reach each time.

While there is still area for improvement all along the aspects of the problem, some of which is also mentioned on the next page, this study made a good step in examining a lot of areas of the problem and emphasizing on what could affect the performance of it the most.

Chapter 7

Future Items

Future improvements on this code and the model can be performed, in order to examine the impact that different stack of LSTM layers can have on the performance of the model.

Alternations like a different CRS score scaling can be tried in to determine if the model can perform better if the CRS scores are not in the (0,1) range but either 0 or 1 individually, making the classifications more binary.

Further improvements on the sequence-to-sequence model, e.g.: use of the attention technique in between the encoder/decoder layers , can also be introduced in order to check if this type of stack set will perform better with the specific dataset.

Other techniques that can be tried by further modifying the algorithms include the biderctional LSTMs or even an approach with SVM (Support Vector Machine) algorithms can be combined, so that the bias of the existing dataset is to be eliminated.

References

Aziza, N. A., Mahadic, R., & Janorb, R. M. (2013). Comparative Departmental Efficiency Analysis within a University: A DEA Approach. *Procedia - Social and Behavioral Science vol. 90*, 540-548.

Bae, K., Ryu, H., & Shin, H. (2019). Does Adam optimizer keep close to the optimal point. *arXiv: Learning*. Retrieved 11 2, 2022, from https://arxiv.org/abs/1911.00289

Ban, H., & Zhong, X. (2022). Pre-trained network-based transfer learning: A small-sample machine learning approach to nuclear power plant classification problem. *Annals of Nuclear Energy*.

Beaufays, F. (n.d.). *The neural networks behind Google Voice transcription*. Retrieved 10 23, 2022, from http://googleresearch.blogspot.co.at/2015/08/the-neural-networks-behind-google-voice.html

Bishop, C. M. (1999). *Pattern Recognition and Feedforward Neural Networks*. Retrieved 9 25, 2022, from https://microsoft.com/en-us/research/publication/pattern-recognition-and-feedforward-neural-networks

Borovkova, S., & Tsiamas, I. (2019). An ensemble of LSTM neural networks for high-frequency stock market classification. *Journal of Forecasting, 38*(6), 600-619. Retrieved 11 3, 2022, from https://onlinelibrary.wiley.com/doi/full/10.1002/for.2585

Bowling, M., Fürnkranz, J., Graepel, T., & Musick, R. (2006). Machine learning and games. *Machine Learning, 63*(3), 211-215. Retrieved 9 26, 2022, from http://ke.tu-darmstadt.de/~juffi/publications/mlj-games-editorial.pdf

Buber, E., & Diri, R. (2018). Performance Analysis and CPU vs GPU Comparison for Deep Learning. *2018 6th International Conference on Control Engineering & Information Technology*, 1-6.

Castillo, E., Cobo, A., Gutiérrez, J. M., & Pruneda, R. E. (1999). *Introduction to Neural Networks*. Retrieved 9 26, 2022, from https://link.springer.com/chapter/10.1007/978-1-4615-5601-5_1

Chakraborttii, C., Sinha, V., & Litz, H. (2018). *SSD QoS Improvements through Machine Learning*. Retrieved 11 3, 2022, from https://doi.org/10.1145/3267809.3275453

Chung, J., Gulcehre, C., Cho, K., Bengio, Y., Bengio, Y., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv: Neural and Evolutionary Computing*. Retrieved 9 25, 2022, from https://arxiv.org/abs/1412.3555

Chung-Cheng etal, C. (2018). State-of-the-Art Speech Recognition with Sequence-to-Sequence Models. *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 4774-4778.

Donahue, J., Hendricks, L. A., Guadarrama, S., Rohrbach, M., Venugopalan, S., Darrell, T., & Saenko, K. (2015). *Long-term recurrent convolutional networks for visual recognition and description*. Retrieved 9 26, 2022, from http://openaccess.thecvf.com/content_cvpr_2015/html/donahue_long-term_recurrent_convolutional_2015_cvpr_paper.html

Gers, F. A., Schraudolph, N. N., & Schmidhuber, J. (2003). Learning precise timing with lstm recurrent networks. *Journal of Machine Learning Research, 3*(1). Retrieved 9 23, 2022, from http://jmlr.org/papers/volume3/gers02a/gers02a.pdf

Heaton, J. (2017). Ian Goodfellow, Yoshua Bengio, and Aaron Courville: Deep learning. *Genetic Programming and Evolvable Machines, 19*(1), 305-307. Retrieved 9 27, 2022, from https://link.springer.com/content/pdf/10.1007/s10710-017-9314-z.pdf

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation, 9*(8), 1735–1780. Retrieved 9 25, 2022

Islam, M. S., Mousumi, S. S., Abujar, S., & Hossain, S. A. (2019). Sequence-to-sequence Bangla Sentence Generation with LSTM Recurrent Neural Networks. *Procedia Computer Science, 152*, 51-58. Retrieved 9 23, 2022, from https://sciencedirect.com/science/article/pii/s1877050919306775

Jiaxuan, Z., & Shunyong, L. (2022). Air quality index forecast in Beijing based on CNN-LSTM multi-model . *Chemosphere vol. 308*.

Karpathy, A., Johnson, J., & Li, F.-F. (2015). Visualizing and Understanding Recurrent Networks. *arXiv: Learning*. Retrieved 9 27, 2022, from https://arxiv.org/abs/1506.02078

Koutroumbas, S., & Theodoridis, K. (2008). *Pattern Recognition - 4th Edition.* Retrieved 9 26, 2022, from https://www.elsevier.com/books/pattern-recognition/theodoridis/978-1-59749-272-0

Krause, B., Murray, I., Renals, S., & Lu, L. (2017). *Multiplicative LSTM for sequence modelling*. Retrieved 9 26, 2022, from https://openreview.net/forum?id=sjcs5rxfl

Liang et al., Z. (2020). Abstractive social media text summarization using selective reinforced. *Neurocomputing*.

Lindsay, R. P. (1964). The Impact of Automation On Public Administration. *Western Political Quarterly*, 78-81.

McQueen, R. J., Garner, S. R., Nevill-Manning, C. G., & Witten, I. H. (1995). Applying machine learning to agricultural data. *Computers and Electronics in Agriculture, 12*(4), 275-293. Retrieved 9 26, 2022, from https://sciencedirect.com/science/article/pii/0168169995986019

Messina, R., & Louradour, J. (2015). *Segmentation-free handwritten Chinese text recognition with LSTM-RNN*. Retrieved 11 3, 2022, from https://dl.acm.org/citation.cfm?id=2880728

Monner, D., & Reggia, J. A. (2012). A generalized LSTM-like training algorithm for second-order recurrent neural networks. *Neural Networks, 25*(1), 70-83. Retrieved 9 23, 2022, from http://overcomplete.net/papers/nn2012.pdf

Nguyen, G. H., Bouzerdoum, A., & Phung, S. L. (2009). *Learning pattern classification tasks with imbalanced data sets*. Retrieved 9 26, 2022, from http://ro.uow.edu.au/cgi/viewcontent.cgi?article=1806&context=infopapers

Pham, H., Manzini, T., Liang, P. P., & Póczos, B. (2018). Seq2Seq2Sentiment: Multimodal Sequence to Sequence Models for Sentiment Analysis. *arXiv: Computation and Language*. Retrieved 9 27, 2022, from https://arxiv.org/pdf/1807.03915

Philipp, G., Song, D., & Carbonell, J. G. (2017). The exploding gradient problem demystified - definition, prevalence, impact, origin, tradeoffs, and solutions. *arXiv: Learning*. Retrieved 11 1, 2022, from https://arxiv.org/pdf/1712.05577

Python.org. (2020). Retrieved from https://www.python.org/downloads/release/python-390/

Ray, A., Rajeswar, S., & Chaudhury, S. (2015). *Text recognition using deep BLSTM networks*. Retrieved 11 3, 2022, from http://ieeexplore.ieee.org/document/7050699

Seiford, L. M. (1994). *A DEA Bibliography (1978–1992)*. Retrieved 9 25, 2022, from https://link.springer.com/chapter/10.1007/978-94-011-0637-5_22

Shafi, I., Ahmad, J., Shah, S. I., & Kashif, F. M. (2006). *Impact of Varying Neurons and Hidden Layers in Neural Network Architecture for a Time Frequency Application*. Retrieved 11 1, 2022, from http://yadda.icm.edu.pl/yadda/element/bwmeta1.element.ieee-000004196403

Shi, X., Chen, Z., Wang, H., Yeung, D.-Y., Wong, W.-k., & Woo, W.-c. (2015). *Convolutional LSTM Network: a machine learning approach for precipitation nowcasting*. Retrieved 11 3, 2022, from https://papers.nips.cc/paper/5955-convolutional-lstm-network-a-machine-learning-approach-for-precipitation-nowcasting.pdf

Sindhu, V. (2020). AN EMPIRICAL SCIENCE RESEARCH ON BIOINFORMATICS IN MACHINE LEARNING. *JOURNAL OF MECHANICS OF CONTINUA AND MATHEMATICAL SCIENCES vol7*.

Thanassoulis, E., & Conceição A. Silva, M. (2018). Measuring Efficiency Through Data Envelopment Analysis. *Impact vol. 2018*, 37-41.

Tsagkaris, K., Katidiotis, A., & Demestichas, P. (2008). Neural network-based learning schemes for cognitive radio systems. *Computer Communications, 31*(14), 3394-3404. Retrieved 11 1, 2022, from https://sciencedirect.com/science/article/pii/s0140366408003393

Varian, H. R. (2014). Big Data: New Tricks for Econometrics. *Journal of Economic Perspectives, 28*(2). Retrieved 9 26, 2022, from http://people.ischool.berkeley.edu/~hal/papers/2013/ml.pdf

Wang, G. (2019). Deep Text-to-Speech System with Seq2Seq Model. *arXiv: Computation and Language*. Retrieved 9 27, 2022, from https://arxiv.org/abs/1903.07398

Wu, S., Falk, T. H., & Chan, W.-Y. (2011). Automatic speech emotion recognition using modulation spectral features. *Speech Communication, 53*(5), 768-785. Retrieved 9 27, 2022, from http://www-gth.die.upm.es/research/documentation/referencias/wu_automatic.pdf

Yıldırım, S., & Asgari-Chenaghlu, M. (2018). *Mastering Transformers.* Birmingham, Mumbai: Packt.

Zhang, B., Wang, Y., & Chen, F. (2014). Multilabel Image Classification Via High-Order Label Correlation Driven Active Learning. *IEEE Transactions on Image Processing, 23*(3), 1430-1441. Retrieved 9 26, 2022, from https://ncbi.nlm.nih.gov/pubmed/24723538