COS711 Assignment 3

Luveshan Marimuthu
u17087709
https://github.com/Luveshanm/cos711Assignment5
Dept. of Computer Science
University of Pretoria

***Abstract*—The purpose of this report is to compare the performance of two deep learning techniques on air quality prediction. The report shows the results of a Convolutional Neural Network (CNN) and a Long Short-Term Memory (LSTM) Network which are implemented in Python using the TensorFlow/Keras API. The data is preprocessed to fill in missing values and scale the input values for each feature (column). I report on the generalization results of different CNN and LSTM configurations and show that the LSTM model performs better than the CNN model.**

***Index Terms*—Convolutional Neural Network, Long Short-Term Memory, Deep Learning, Air Quality.**

I. Introduction

In this report I apply two deep learning techniques, Convolutional Neural Network(CNN) and Long Short-Term Memory(LSTM) Network, to create an air quality prediction model. The data pre-processing as well as the prediction models are implemented in python.

The data set used to train and test the models is provided by https://zindi.africa which is an African data science and machine learning competition platform. Each pattern in the data set is a 5-day series of hourly weather data readings from different sensors across Uganda. Therefore the air quality prediction task is of a temporal nature. The target value is the air quality level at exactly 24 hours after the last sensor readings. The air quality is measured in PM2.5 (particulate matter smaller than 2.5 micrometers in diameter or around 1/30th the thickness of a human hair) and is measured in micro-grams per cubic meter (μ/m3). [1]

The pre-processing of the data set involves converting the column values into numerical arrays, filling in the missing values for some columns in each row in the data and scaling all of the values to a common range in preparation for machine learning.

Concolutional Neural Networks(CNNs) is a category of deep neural networks that is typically used for image classification. CNNS have however also been applied to time series prediction tasks. In this assignment a convolutional neural network is implemented to train on the data from the weather sensors as 2D arrays of readings for different 'features' (columns). Given the structure of the data, the 2D inputs are matrices of *features x readings*.

Long Short-Term Memory(LSTM) Networks are suited for making predictions on time series data and is able to deal with the vanishing gradient problem of traditional Recurrent Neural Networks (RNN). The LSTM model is trained on the same data as the CNN with input patterns being matrices of *features x readings*.

For this particular problem the results show that, while the CNN did not fall too far behind, the LSTM model is better suited to air quality prediction using temporal input data. Different configurations were tested and adding dropout regularization could be an option for practical use and generalizability.

II. Background

The weather data needs to be analyzed and pre-processed to make it possible for the deep learning algorithms to more effectively learn the hidden relationships between the input features and target values. Input scaling is a crucial part of pre-processing as having features on a similar scale can help the models converge more quickly.

In the given data set, each column consisted of a string of values. These values were converted to numerical arrays to create a 2D array representation on the data. As with most real world machine learning problems, the data had missing entries scattered throughout the set. Usually the rows with missing values are removed however there were too many rows that had missing values and so removing these would mean losing somewhat valuable data patterns. Instead the data was filled in by replacing them with an average of the present values. The average was calculated across the array of values in that particular column for that particular row. This was done because the array of values for one row in one specific column are weather readings of 5 consecutive days which, in the literal sense, by nature are more likely to have similar readings because the weather will be somewhat, but not necessarily, similar.

Lastly the values in each column were scaled using standardization (z-score normalization), this time taking into account all of the values across all the rows in the data set.

In the case of CNN, the model will seek to find any spacial relationships between the weather readings. Although the data is temporal in nature, weather readings of 5 consecutive days have a good chance of having similar properties. Furthermore, the CNNs pooling technique for feature extraction could possibly find relationships that generalize across the different sensor locations.

In the case of LSTM, these models excel at time series prediction and I expect it to outperform the CNN, together with the fact that CNNs were designed more for image analysis.

LSTMs are able to 'remember' past data patterns which should prove useful in this task. By remembering more information the model could simulate having a longer series per pattern during training, for example hypothetically having 7 days worth of readings instead of only the 5 days provided in each pattern.
The various hyper-parameters and number of layers (depth) of each model was decided through trail and error. Initially, the models began more simple (smaller) and increased in complexity (number of layers or units per layer) until they reached diminishing return or the training time became too long. The activation functions and optimizers play an important role in training and were chosen through their known successes in practice.
The metric used by Zindi to judge the competition entries was the Root Mean Squared Error function.[1] For this reason, I used the Mean Squared Error loss function and included the Root Mean Squared Error metric in the evaluation to determine the relative performance/'accuracy' of my models.
Note: I did not enter the competition.

III. Experimental Set-Up

The deep learning models discussed below were implemented in Python using the following libraries: TensorFlow, Keras, pandas, numpy and scikit-learn.
In order to reduce the amount of time it takes to train both models, I used the GPU version of tensorflow with a Nvidia GPU. This version makes use of the GPU-accelerated NVIDIA CUDA® Deep Neural Network library (cuDNN) [2]. An installation guide can be found at https://www.tensorflow.org/install/gpu.

A. Pre-processing

The data was loaded from the .csv file into a pandas dataframe. As mentioned earlier the column values were strings of values so these had to be converted into 1D arrays of floating point values. Afterwards, each column was separated into individual 2D arrays to make the following pre-processing steps easier to compute.
Note: After pre-processing is complete, a new .csv file is constructed and is used by the deep learning algorithms.

1) Missing Values : Missing values were scattered all throughout the data set. Some were singular missing entries and others were long strings of missing entries, probably due to events such as power cuts or maintenance time. In light of this, two strategies were used to fill in the missing entries:

First, the singular missing values were found and replaced with the average of the values immediately before and after the missing value. In terms of weather readings this was ideal because even if the before and after readings are very different, the change from one to the other is gradual thus using the average as the inbetween value. The corner cases are either the very first or the very last value is missing, in which case it is simply filled in with the value immediately after or before respectively.
Second, some rows/columns still contain strings of two or more consecutive missing values. In this case the missing entries are replaced with an average of all the values of that particular column for that particular row. As explained in the background, this was done because the array of values for one row in one specific column are weather readings of 5 consecutive days which in the literal sense by nature are more likely to have similar readings because the weather will be somewhat, but not necessarily, similar. Furthermore, since no date and time information was given with the data, the 5-day readings across the rows could possibly be months apart and have no correlation in terms of weather which makes calculating the average across the rows of a specific column inappropriate for this scenario.

2) Input Scaling : The individual column arrays were scaled using standardization (z-score normalization) on all of the values across all the rows in the data set for each column. MinMax scaling is not appropriate in this case because the values are taken from sensors so the true minimum and maximum values are not truly represented in the data set. The standardization method used the mean and standard deviation of each column to scale all the values in that column to the range *(-1, 1)*. The same input scaling was used for the CNN and LSTM.

B. CNN

The Convolutional Neural Network was implemented in Python using the Tensorflow/Keras Sequential model. I loaded the new pre-processed data set and converted the data into a 3D numpy array i.e *rows x features x values*. The dimensions of the full data is (15539, 6, 121) and the input shape specified in the model is (6, 121).

1) Hyper-parameters: The CNN consists of two 1D convolutional layers with each layer having a 1D Max Pooling layer and two dense (fully connected) layers. I chose to use two convolutional layers because having just one would not be enough to effectively train on the data given the number of values per pattern however having more than two convolutional layers did not show a significant increase in accuracy/decrease in loss value. The first fully connected layer flattens the convolutional layer output into a 1D array and the second fully connected layer is the output layer that makes the prediction. I decided on Max Pooling because it is the most common approach to pooling layers in CNNs.
The number of units in each layer was decided through trail and error. The two convolutional layers have 128 units each, the first fully connected layer has 64 units and the output layer has 1 unit. Due to the input dimensions, specifically the fact that there are only 6 rows, the values that worked for the kernel and pool sizes are respectively 3 and 2 for the first convolutional layer and 1 and 1 for the second convolutional layer.
The activation functions were chosen based on suitability.

Since this is a regression task, the output layer uses a linear activation function instead of softmax which is more suited to classification. All layers initially used the ReLU activation function for its low computational cost and faster convergence however after experimenting with the LSTM in the next section I changed the activation of the convolutional layers to tanh because it showed better performance.

2) Training: The CNN is trained using the Adam optimizer with a learning rate of 0.001 and Mean Squared Error loss function. The Adam optimizer was used because it is the most common and better performing optimizer available. The Mean Squared Error loss function was chosen because the official competition on https://zindi.africa was evaluated using the Root Mean Squared Error metric [1], which is also included in the evaluation output of the CNN.
I decided to add dropout regularization to the CNN in an attempt to better its generalization. Each layer uses a dropout rate of 20%.
During training the CNN uses a batch size of 32, the data patterns are shuffled on every epoch and is trained for 25 epochs in total. More than 25 epochs did not show any significant increase in generalizability. The data set was split into an 80% training and 20% validation set.

C. LSTM

The Long Short-Term Memory Network was also implemented in Python using the Tensorflow/Keras Sequential model. I loaded the new pre-processed data set and converted the data into a 3D numpy array i.e *rows x features x values*. The dimensions of the full data is (15539, 6, 121) and the input shape specified in the models is (6, 121).

1) Hyper-parameters: The LSTM consists of two LSTM layers and two dense (fully connected) layers. Through trial and error, adding or removing either LSTM or dense layers did not lead to better performance. The first fully connected layer flattens the LSTM layer output into a 1D array and the second fully connected layer is the output layer that makes the prediction.
The number of units in each layer were also decided through trail and error. The first LSTM layer has 128 units, the second LSTM layer has 64 units, the first fully connected layer has 64 units and lastly the output layer has 1 unit.
The activation functions were chosen based on suitability. Since this is a regression task, the output layer uses a linear activation function instead of softmax which is more suited to classification. The two LSTM layers use the tanh activation function since it is better optimized to run on a GPU which significantly decreased training time. I noticed the increase in performance as well and decided to try tanh on the CNN.

2) Training: The LSTM is also trained using the Adam optimizer with a learning rate of 0.001 and Mean Squared Error loss function. The Adam optimizer was used because it is the most common and better performing optimizer available. The Mean Squared Error loss function was chosen because the official competition on https://zindi.africa was evaluated using the Root Mean Squared Error metric [1], which is also included in the evaluation output of the CNN.
I decided to add dropout regularization to the LSTM model in an attempt to better its generalization. Each layer uses a dropout rate of 20%.
During training the LSTM uses a batch size of 32, the data patterns are shuffled on every epoch and is trained for 25 epochs in total. More than 25 epochs did not show any significant increase in generalizability. The data set was split into an 80% training and 20% validation set.

IV. Research Results

For each deep learning model, five runs, each over 25 epochs, were taken and the averages and standard deviations were calculated. The tables below show the Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) for the training set (T_) and validation set (V_). Most of the higher ranking RMSE results on the competition leaderboard were in the lower 30s so I used that as a point of reference.
Note: The values shown on the leaderboard are for the test set provided by Zindi for which the target values are not known. Therefore the RMSE values for the models below will most likely be somewhat higher if used on said test set.

TABLE I
CNN Performance (ReLU activation)

	1	2	3	4	5	Avg	std
T_MSE	103.60	109.20	123.23	99.31	103.85	**107.84**	**8.31**
T_RMSE	10.18	10.45	11.10	9.97	10.19	**10.38**	**0.39**
V_MSE	1003.25	982.51	979.38	1001.64	969.80	**987.32**	**13.05**
V_RMSE	31.73	31.47	31.43	31.60	31.27	**31.5**	**0.16**

Table I shows the performance of the Convolutional Neural Network using the ReLU activation function. The loss values of the validation(test) set were much higher than that of the training set however the RMSE values for the validation set were already at the lower 30s so the CNN is off to a good start. The standard deviation of the RMSE values for the validation set is very low meaning the results were fairly consistent. Since this is the first models results, the aim of the results to follow will simply be to find better performing models or configurations.

TABLE II
LSTM Performance (tanh activation)

	1	2	3	4	5	Avg	std
T_MSE	38.96	38.20	41.47	41.49	37.03	**39.43**	**1.78**
T_RMSE	6.24	6.18	6.44	6.44	6.09	**6.28**	**0.14**
V_MSE	804.34	811.83	843.77	830.21	828.18	**823.66**	**14.0**
V_RMS	28.48	28.61	29.16	28.94	28.89	**28.82**	**0.24**

Table II shows the performance of the Long Short-Term Memory network using the tanh activation function. Again the loss values of the validation(test) set were much higher than that of the training set but not as high as that of

the CNN above, 163.66 lower on average, indicating better generalization. The main focus in this model however is the much lower RMSE values on the validation set with an average of 28.82 and standard deviation of 0.24. Since deep learning with LSTMs excel at time series prediction, these results were moderately expected.

TABLE III
CNN PERFORMANCE (TANH ACTIVATION)

	1	2	3	4	5	Avg	std
T_MSE	59.12	59.03	51.85	66.23	63.59	**59.96**	**4.9**
T_RMSE	7.69	7.69	7.20	8.13	7.97	**7.74**	**0.32**
V_MSE	987.21	996.30	956.63	923.03	980.36	**968.71**	**26.2**
V_RMS	31.54	31.65	31.06	30.49	31.45	**31.24**	**0.42**

Looking at the results of the LSTM model, I replaced the ReLU activation function in the convolutional layers of the CNN with the tanh activation. Looking at table III, the change made a massive improvement to the models performance on the training data with an over 40% decrease in the average loss value (from 107.84 to 59.96). Unfortunately, it made no difference at all to the performance on the validation set, which is an indication of it simply over-fitting the data.

In light of the over-fitting I decided to try dropout regularization to improve both models generalization.

TABLE IV
LSTM PERFORMANCE (WITH DROPOUT)

	1	2	3	4	5	Avg	std
T_MSE	196.12	198.93	204.53	236.38	211.41	**209.47**	**14.43**
T_RMSE	14.0	14.01	14.29	15.38	14.54	**14.44**	**0.51**
V_MSE	872.36	861.38	840.05	853.14	831.52	**851.69**	**14.6**
V_RMS	29.67	29.45	29.11	29.32	28.96	**29.30**	**0.25**

Table IV shows the performance of the LSTM from table II this time with dropout regularization at each layer. The training set MSE and RMSE values increased, showing possibly less over-fitting, but the performance on the validation set did not improve. Even though the generalization did not improve, the difference is fairly negligible, 0.48 increase in RMSE on average, and there is no evidence to say that it will not (slightly) outperform the earlier model for future data sets/air quality predictions.

TABLE V
CNN PERFORMANCE (WITH DROPOUT)

	1	2	3	4	5	Avg	std
T_MSE	368.0	365.11	371.70	358.84	377.21	**368.17**	**6.17**
T_RMSE	19.19	19.11	19.28	18.94	19.43	**19.19**	**0.16**
V_MSE	1013.79	996.36	1014.50	1000.97	1013.21	**1007.77**	**7.58**
V_RMS	31.98	31.70	32.0	31.76	31.97	**31.88**	**0.13**

Table V shows the performance of the CNN from table III this time with dropout regularization at each layer. The effect of adding the dropout regularization is exactly the same as its effect on the LSTM in table IV. The MSE and RMSE values for the training set increased when compared to the CNN results in table I and III. The performance on the validation set did not improve much however the difference is fairly negligible. In this case 0.38 average increase from table I and 0.64 average increase from table III.

V. CONCLUSIONS

In conclusion, two deep learning models were trained on air quality prediction using real world data provided by https://zindi.africa. The performance of these models, a Convolutional Neural Network(CNN) and Long Short-Term Memory network(LSTM), were compared to determine which technique is better at predicting the air quality using past weather readings. This report shows that Long Short-Term Memory is the more suitable deep learning technique for this problem. Overall, the LSTMS's RMSE and loss(MSE) values were lower than the CNN's values indicating better prediction capability. More specifically, the model shown in table II outperformed all of the other models tested in the report. While on paper the results are clear, the LSTM implemented with dropout seems to have potential for high performance in practice. From this assignment I have learnt that deep learning is a great improvement to the original concept of machine learning and when optimized and applied to the appropriate situations can be used to more efficiently perform tasks that can otherwise only be performed by the brain.

REFERENCES

[1] *AirQo Ugandan Air Quality Forecast Challenge*. 2020. URL: https://zindi.africa/competitions/airqo-ugandan-air-quality-forecast-challenge/data (visited on 06/10/2020).

[2] *NVIDIA cuDNN*. 2020. URL: https://developer.nvidia.com/cudnn (visited on 06/10/2020).