INNS Coursework

Module Code: COM00029H
Exam Number: Y3856226

1 Discussion of architectures

The Dataset

The Seoul Bike dataset shows the demand for a bike rental scheme in Seoul, across 365 days, with one sample per hour for a total dataset of 8760 samples. The dependent variable is the 'Rented Bike Count' which is the measure of the number of bikes rented each hour. As a continuous value is being predicted, a forecast of the number of bikes rented in a given hour, then this is a regression problem. The dataset contains a wide range of independent variables, including; weather data (e.g. temperature, humidity, and rainfall), time data (date and hour), and miscellaneous factors (season, if the scheme is functioning, and if it is a national holiday). Initially, it was considered that the dataset may be too small for training, due to the wide amount of variance caused by the different parameters (i.e. weather, holidays, and day of the week), however testing proved that this wasn't a problem.

Potential Networks

One of the most basic Neural Networks (NN) that may be able to solve this problem is a Multilayer Perceptron (MLP). An MLP is a simple feedforward network composed of 3 or more layers; an input layer, at least one hidden layer, and an output layer, each with an activation function. As the most simple solution, this should be tested as a baseline to compare other network models against.

As we are working with time-series data, an Recurrent Neural Network (RNN) should also be considered. An RNN is a network in which data is passed from each hidden node back into every hidden node, therefore previous time steps influence the output of the current step. There are three types of RNN to consider, which can all be seen in Figure 1 [1]: Simple RNN, Gated Recurrent Unit (GRU), and Long Short Term Memory (LSTM). A Simple RNN will not be tested, as the simple structure means that information fades away quickly over time which may reduce the effect of the advantages an RNN provides, while both GRU and LSTM are able to retain information over a longer period due to the use of a series of gates. This can be extremely useful as it allows more complex connections to be formed, for example it may learn that if it rained in the morning then fewer people want to rent a bike in the afternoon, even though it is sunny. Both architectures take different approaches: GRU uses a simple gate structure meaning it performs quicker, while LSTMs use a more complex structure that may perform better, providing system resource limitations aren't a factor. Therefore, both may be suitable and shall be tested.

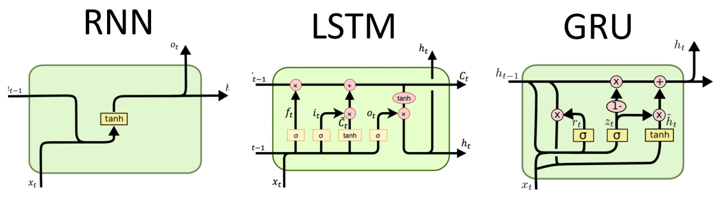

Figure 1: RNN Architectures [1].

Convolutional Neural Networks (CNNs) are a variation of MLPs, consisting of a varying amount of 3 key layers: Convolution layers, Pooling layers, and a Fully Connected layer. A typical CNN can be seen in Figure 2. Convolution Layers use kernels as feature detectors, with extracted features being the input to the next layer. Pooling layers are designed to downsample, consuming small chunks (typically 2 by 2) and aggregating them into a single value, typically max-pooling is used but other schemes such as a weighted average or L2 norm can be used. Finally, after a series of Convolution and Pooling layers, the output is fed into an MLP. CNNs are particularly good at extracting features, meaning feature extraction does not have to be carried out manually, which may be usefu as there is little prior knowledge of the dataset or problem.

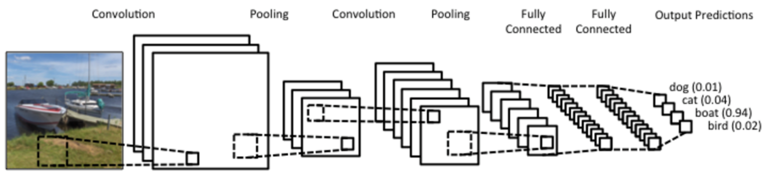

Figure 2: CNN architecture.

To fairly test the different networks, similar parameters and architectures were used during initial testing. Most architectures consisted of a 14 unit input layer: for the ANN architecture this was a relu dense layer, for the LSTM architecture an LSTM layer, for the GRU architecture a GRU layer. The CNN had to be structured differently and consisted of a 5 by 3 input (with padding added to allow the shape), passed into 2 2D convolution layers and a max-pooling layer. All the networks then consisted of; a 6 unit dense layer, dropout

	ANN	CNN	LSTM	GRU
Root Mean-Square Error	37.97	25.88	119.70	143.50
R-Squared	0.996	0.998	0.966	0.951

Table 1: Architecture Testing Results

layer, and a single unit dense output layer. The Adam optimizer, which is a combination of RMSprop and Stochastic Gradient Descent with momentum, was used due to its speed and accuracy. The networks were then all run with a batch size of 24 (as the data contained 24 hour days) and 500 epochs, however, a callback was implemented to stop this early if the Mean Square Error (MSE) did not improve within 10 epochs.

Table 1 shows the results from testing the 4 proposed networks. All networks perform well, with a high R-Squared (R2) score throughout, showing that all models fit the data well, with the CNN having the highest R2 score and lowest RMSE. The comparatively poor performance of the LSTM and GRU architectures was surprising due to their suitability to time series data, and the difference can be seen clearly across Figures 3 to 5. However, the CNNs performance is likely due to its ability to extract features, combined with the lack of manual feature extraction for other architectures. Therefore, due to the high performance of the CNN to extract features combined with a fully connected layer, this shall be focused on in future testing. I shall also test if performance can be increased by adding an LSTM layer after the CNN, as the benefit of the CNNs feature extraction may benefit the LSTM.

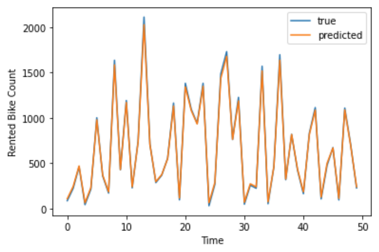

Figure 3: CNN Rented Bike Count Predictions

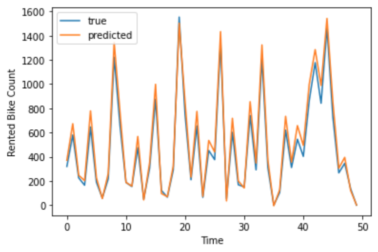

Figure 4: LSTM Rented Bike Count Predictions.

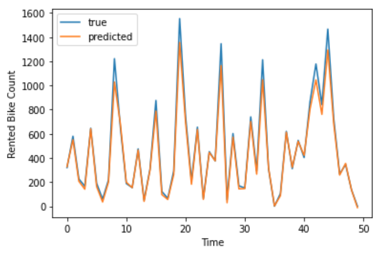

Figure 5: GRU Rented Bike Count Predictions

2 Creation and application of neural networks

Preparation

When investigating the dataset, one of the most interesting fields is Date. There is potentially a lot of information to be learnt from the date; the season, the month, if it's a holiday and if it's a weekend. However, a lot of this information already exists in the holiday and seasons fields, while a new day field shall be created to provide data like if it is a weekend. A month field was also considered, however I believe any important information this could provide is already covered within available fields. The process of removing date and adding

Fields		
Day	Dew Point Temperature	Visibility
Hour	Temperature	Solar Radiation
Holiday	Humidity	Rainfall
Functioning Day	Wind Speed	Snowfall
Season		

Table 2: Fields used from Seoul Bike data.

day will hopefully retain all available information, while reducing complexity and improving performance. Therefore, by dropping the date we remove some duplicate data and simplify the data for the Neural Network. All other columns, see Table 2, shall be kept as they have been deemed important and correlate well with Rented Bike Count.

The next step is to clean the data. I would usually suggest starting by removing any rows containing infinite or null values but there are none in this dataset, however this should be reviewed with any future data. Aside from this, any categorical data (Functioning Day, Holiday, Seasons, and Days) must be encoded into numerical values. This can either be done through an in-built label or one-hot-encoder or simply by replacing values. For example, for Seasons; replace Winter with 0, Spring with 1, Summer with 2 and Autumn with 3 and for Holiday; replace No Holiday with 0 and Holiday with 1.

As discussed by Rodrigues et al [2], most CNNs perform better over non-preprocessed data when trained from scratch, therefore as a CNN is being used for feature extraction, no further pre-processing shall take place. Scaling the data was considered, however initial testing proved that this had no positive impact on performance.

Finally, the data is split into training, testing, and validation data, with 72% being used for training, 20% for testing, and 8% for validation. This is important as it allows metrics (i.e. MSE) to be tested each epoch against the unseen validation data, while the testing data can be given to the network after testing to forecast values which can be used to measure performance with metrics such as R2 and RMSE.

Further Architecture Testing

Due to the high performance of CNNs, two further CNN architectures shall be tested. The first shall be a CNN with a simple fully connected structure, allowing to test if any improvements can be made by modifying the number of layers, number of neurons, or any other parameters. Alongside this, a CNN chained with an LSTM and a fully connected structure shall be tested. As the CNN performed well previously, the CNN architecture shall not be changed, and the focus shall be on modifying the LSTM, the fully connected layers, and tweaking parameters.

	Initial Parameters	CNN Parameters	CNN-LSTM Parameters
Convolution Filters	2	2	6
Dropout Probability	0.1	0.05	0.05
LSTM Units	12	-	8
Hidden Layers	1	1	1
Hidden Units	6	4	6
RMSE	25.88	21.7	11.5
R2	0.980	0.9988	0.9997

Table 3: Final Architecture Parameters and Testing

The input for the CNN shall be a 5 by 3 array, therefore for this to occur, an extra column is added to the dataset called 'fill' consisting of all 0s to avoid skewing the data. The CNN itself shall initially consist of 2 2D convolution layers, both with added padding and a relu activation function. This is then fed into a max-pooling layer, and dropout layer, before being passed into a flattening layer so it is suitable for the following fully connected or LSTM layers.

The basic CNN architecture, with a simple fully-connected layer, consists of a 6 neuron dense layer, dropout layer, and then single neuron output linear layer. This simple architecture performs well, however it will be tested with alternative numbers of layers, neurons, and different parameters in an attempt to improve performance.

The CNN-LSTM is similar, consisting of the same CNN, but a reshaping layer is also required to ensure the data is of the right shape for the RNN. This is then fed into an LSTM layer, with recurrent dropout, before being passed into a fully-connected network with a 6 neuron dense layer, dropout layer, and single neuron linear output layer. This architecture will test if the combined CNN-LSTM improves performance, and providing it does a range of parameters will be tested to maximise performance.

To ensure tests were fair for both architectures, each test was run using the same dataset and repeated 3 times with the average R2 and RMSE values being compared. Initially, different numbers of hidden layers were tested, however, it was found that no matter the number of hidden units this negatively impacted performance, which was probably caused by overfitting. Alongside this, it was found that having 2 dropout layers negatively impacted performance, therefore the dropout layer in the fully connected layer was removed, with the CNN dropout and LSTM dropout remaining. No other significant results occurred, other than slight improvements by reducing the LSTM and hidden units respectively. The best performing results can be seen in Table 3, with comparison to initial CNN parameters from Part 1. Both networks saw a reasonable increase in performance, however, the CNN-LSTM outperformed the more standard CNN and shall be looked at further.

3 Results and Evaluation

Results and Metrics

The core metric used when comparing networks has been the R2 score, the coefficient of determination, which demonstrates the proportion of variance in the dependent variable that is predictable from the independent variables. In practice, this is a value between 0 and 1 showing the percentage of correlation, the results of the CNN-LSTM model produced demonstrated a 99.97% or almost perfect correlation.

Alongside this, MSE was used, which is the average of the square of the errors. This is a good metric as it allows us to measure the real difference between the observed and predicted values, and due to the fact it is squared positive and negative values do not cancel out. The use of MSE in measuring the improvement of the network over time can be seen in Figure 6, demonstrating the error in the network over a series of Epochs. RMSE has also been used as it is a more interpretable statistic than MSE, due to having the same magnitude as plotted data.

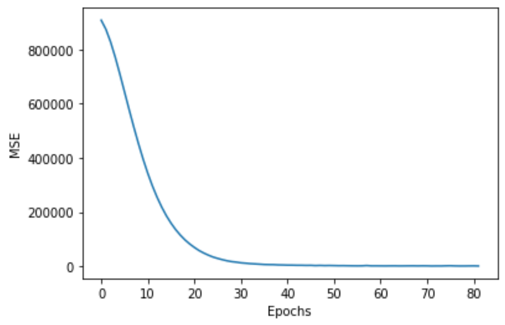

Figure 6: Mean Square Error over training epochs

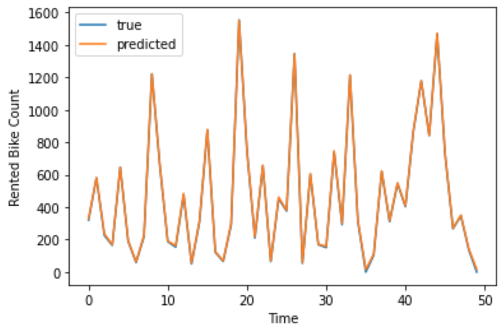

Figure 7: Rented Bike Count True predicitions.

Initially, it was surprising that the LSTM and GRU both had relatively poor performance in comparison to the CNN and ANN. However, it is clear from the results of the chained CNN-LSTM, that this is likely due to the lack of manual feature extraction for the LSTM and GRU architectures. This demonstrates the CNNs ability to extract features, which is particularly useful in circumstances such as these where there is no prior knowledge of the data or problem.

The positive influence of the LSTM is also clear from results, with performance improving compared with the basic CNN. This is likely due to the LSTMs strength in using its long term memory to improve predictions, which must have previously been hampered by unimportant features.

	R2	MSE	RMSE	MAE	Max Error
CNN-LSTM	0.9995	226.81	15.06	12.19	75
Decision Tree	0.9986	32,439	180.11	137.76	1185.06
K-Nearest Neighbours	0.9991	685.92	26.19	22.49	91
Support Vector Machine	-1.1285	890,833	943.84	687.25	3309

Table 4: Results from the best CNN-LSTM run

The final results were very good, with an R2 score of 0.9995 and RMSE of 15.06, seen in Table 4. The RMSE is a particularly good score given the average Rented Bike Count was 705, therefore this is only an error of around 2.1%. This can be seen in a more readable way in Figure 7, with there only being very small differences between the true and predicted values across the first 50 time steps. This demonstrates the trained networks' successful ability to predict the rented bike count to a high standard, outperforming the K-Nearest Neighbors, Support Vector Machines and Decision Tree classifiers.

However, the network is not resilient to other external factors or disruptions that do not render the service non-functional, such as a selection of bikes being broken, difficulty accessing bikes due to road/construction work, or disruptions to public transport. If this occurs, the network would not be able to accurately forecast demand.

4 Further application

The use of GPS data to predict modes of transport presents a very different type of problem, being a classification problem rather than a regression problem. This brings different challenges than the previous dataset, requiring different outputs, activation functions, and an alternative loss function.

One of the largest challenges that may be faced is bias in the data, as environmental issues will have a huge influence on the data. For example, you may be able to cycle much faster along a road if there is a traffic jam, but if there isn't you'd be able to drive much faster, therefore this must either be recorded or data clean of environmental factors will be required.

Etemad et al. [3] proposed a 5 steps framework to predict transportation modes, involving: data preparation, point features generation, trajectory features extraction, noise removal, and normalization. They showed that extracting new features such as bearing rate enabled much higher classifier accuracy (96.5%). However, as shown by Dabiri [4] manual feature extraction has some major drawbacks, by adding human bias in creating efficient features and vulnerability to traffic and environmental conditions. Therefore, due to the improvement in the performance of our LSTM when chained with a CNN and good CNN performance on GPS data being shown by Dabiri [4], I propose that a CNN is used rather than feature extraction.

The network shown to have the highest performance by Dabiri [4], was of a similar structure to our original CNN but with a greater number of layers, likely due to the increased complexity of the dataset. The CNN consisted of 3 repeating sets of units consisting of 2 convolutional layers, followed by a max-pooling layer, this was then fed into 2 pairs of dropout and fully connected layers with the final fully connected layer being the output layer. Therefore, I believe this is a good starting point for the network, from which tests can be carried out using our dataset with different modifications to the architecture.

While using a CNN may be the primary architecture due to its historic performance, it is also important to test different architectures. As an alternative to a CNN, an autoencoder should be tested, while they are primarily used for dimensionality reduction, this means that they can also be useful in feature extraction and classification. Therefore, autoencoders should be tested either independently with fully connected layers or in tandem with a CNN. Alongside this, due to proven performance in our tests on the Seoul Bike data, I believe that adding an LSTM after the CNN may see a marked performance improvement. This is due to the influence of historic data in LSTMs, which may be able to partially mitigate the issue proposed above where traffic may lead to misclassification, as knowledge of the previous activity would influence outputs.

I would recommend the evaluation process stays largely the same, splitting the data to test and validate the network, with each test being repeated 3 times to ensure consistency. However, a new range of metrics must be used. First of all, a new loss function would be required, for this, I recommend Categorical Crossentropy, this is a loss function specialized at multi-class classification tasks in which the model will predict which of the categories is most likely from the data. Alongside this, new evaluation metrics would be required, for this, I recommend measuring the true positive, true negative, false positive, and false negative rates. These can be used to produce a confusion matrix as an easy way to compare network performance, and useful metrics, such as; accuracy, precision, recall, and f1 score.

References

[1] dProgrammer, "Rnn, lstm and gru," [Online]. Available: `http://dprogrammer.org/rnn-lstm-gru`.

[2] L. F. Rodrigues, M. C. Naldi, and J. F. Mari, "Comparing convolutional neural networks and preprocessing techniques for hep-2 cell classification in immunofluorescence images," *Computers in Biology and Medicine*, vol. 116, p. 103 542, 2020.

[3] M. Etemad, A. Soares Júnior, and S. Matwin, "Predicting transportation modes of gps trajectories using feature engineering and noise removal," in *Advances in Artificial Intelligence*, 2018, pp. 259–264.

[4] S. Dabiri and K. Heaslip, "Inferring transportation modes from gps trajectories using a convolutional neural network," *Transportation Research Part C: Emerging Technologies*, vol. 86, pp. 360–371, 2018.