Introduction to Neural Networks Open Individual Assessment

Exam Number: Y3868343

Discussion of Architectures

The scenario for this assessment is the introduction of a rental bike scheme. It is important to have the right number of rental bikes available and accessible to the public at the right time to reduce waiting time. An essential part of this is the prediction of bike count required at each hour of the day to ensure a stable supply of rental bikes.

The data for this assessment are taken from the UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Seoul+Bike+Sharing+Demand)

The data are for each hour of each day, and relate to weather information (Temperature, Humidity, Windspeed, Visibility, Dewpoint, Solar radiation, Snowfall, Rainfall), and the number of bikes rented per hour. In total, we are given 8760 records of data provided by a csv file with the following fields:

- Date (year-month-day)
- **Rented Bike count (Integer)**
- Hour (Integer)
- Temperature (Float)
- Humidity /% (Integer)
- Wind Speed /ms^{-1} (Float)
- Visibility /10m (Integer)
- Dew Point temperature /°C (Float)
- Solar Radiation /MJm^{-2}
- Rainfall /mm (Float)
- Snowfall /cm (Float)
- Season (‘Winter’, ‘Spring’, ‘Summer’, ‘Autumn’)
- Holiday (‘Holiday’, ‘No Holiday’)
- Functioning Day (‘Yes’, ‘No’)

We will be building and training a neural network model to predict the Rented Bike count, so the network will act as a simple regression model given the other fields as inputs. At first glance, this may seem to be a simple regression model which could be solved using a simple feedforward neural network or even a basic Multi-Layer Perceptron, but the inclusion of time fields implies the data could be considered sequential. This means that it might be more suitable to opt for a deep neural model such as a recurrent neural network which includes delayed outputs from a previous cycle of processing as additional inputs of the network. In this sense, a recurrent neural network’s decision process is influenced by previous processes and acts as a kind of memory for the network. Data values are listed in the csv file chronologically with equal timesteps (one recording every hour) which would be suitable for a recurrent architecture since the number of bikes rented in previous hours are likely to influence the number of bikes rented in the current hour.

Using MATLABs default training parameters and hidden layers of 10 neurons, I compared the performance of a feedforward network and a recurrent network with a delay of 24 (to match up hours in a day). The recurrent network vastly outperformed the feedforward network with a mean squared error of 2.77e-6 and a testing R value

of 0.97 compared to a mean squared error of 1.91e-7 and a testing R value of 0.79. With this initial information, it is clear to see that the recurrent architecture would be the optimal choice.

Creation and Application

Since we are designing a recurrent neural network, timesteps are implied through the data order itself so all generic time-based fields are removed from the table (Date, Hour, Seasons). The only time-related field kept for the network is Holiday as this is independent time and occurs all year round. Some might argue that the time of year would influence the bikes rented due to seasonal weather changes so Date should remain included, but I have decided against this as these weather factors would be represented appropriately by the other fields and therefore has no use. We are also told through the data whether a certain day contains functioning hours (Functioning Day) i.e. days that are non-functioning will have rented zero bikes. Since these records are useless to the network since these tell us no new information, I have decided to omit them from training as well as removing the Functioning Day field.

We can then extract the inputs and outputs from the remaining fields. We know that we are creating a network to predict the number of bikes rented on an hourly basis, so we separate out the Rented Bike Count field as our network's output and leave the remaining environmental factors (Temperature, Humidity, Wind Speed, Visibility, Dew Point Temperature, Solar Radiation, Rainfall, Snowfall) as the network's inputs. All that is left to do is normalise the data values bilaterally in the range of [-1,1] as well as binarizing the Holiday field to 1s for Holiday and 0s for no holiday. The MATLAB code to replicate the data pre-processing discussed above is shown below:

```
X = SeoulBikeData;

X(X.FunctioningDay == "No",:) = []; %remove non functioning records
Y = normc(table2array(X(:,2))); %get rented bike count
X.Holiday = X.Holiday == "Holiday"; %binarise holiday field
X(:,[1,2,3,12,14])=[]; %remove useless fields
X = normc(table2array(X));
```

With the data extracted and normalised we can then design and train the architecture of the network using MATLABs inbuilt neural network toolbox by running `>> nnstart`. Since we want to create a recurrent neural network, we choose the dynamic time series app and select the Nonlinear Autoregressive with External Input (NARX) solution. We can then simply select the pre-processed arrays of our inputs and outputs as well as what proportions to divide the training/testing/validating sets into. For the purposes of this solution, we will simply choose a default 70/15/15 ratio.

The next processing stage allows us to fine tune the parameters of the network such as number of neurons in the hidden layer and the chosen delay for the recurrence. Since we know that data cycles through a 24hr period and repeats, I have decided to choose 24 as the delay. Records created at the same time in separate days are likely to share some similarities. Preferably, we would want to input outputs from the same hour of the same weekday in the previous week, but the MATLAB training toolbox

restricts delay values to the range [1,100] and integrating the week delay feature would require a value of 168.

Choosing the number of neurons cannot be gauged by inspecting the data alone so this needs to be done through experimentation. Networks with hidden layers of sizes 4 through 20 are then trained using the faster Levenberg-Marquardt method to be compared and evaluated. Each network is trained repeatedly to reduce the effect of any possible outliers. Below shows a sample of results from these experiments:

	4	6	8	10	12	14	16	18	20
MSE /e-6	3.04	2.57	2.63	2.60	2.36	2.64	3.02	2.82	2.74
R value	0.975	0.974	0.971	0.975	0.977	0.973	0.972	0.972	0.975

If we choose too low a number, we risk not producing an accurate enough solution and if we choose too great a number, we risk overfitting the data which will in turn also produce an unsuitable solution. From the table, it is difficult to infer a clear optimal choice mostly due to the variation in network initialisation and training but, of our samples, we can see that using 12 neurons appears to provide strong values for both the mean square error and the R value. As an additional measure, I tested a large number of neurons (30) to ensure my selected range was not an underestimate which produced undesirable results i.e. MSE > 3e-6 and R < 0.965.

Now that we have a finalised the structure of the network, we can now investigate the effect of alternative training methods and techniques. Within the MATLAB toolbox we have the option to choose between Levenberg-Marquardt, Bayesian Regularisation, and Scaled Conjugate Gradient training algorithms. Levenberg-Marquardt typically takes less time to train but requires more memory which makes it good for experimenting with and testing model parameters. During an example training session, this method produced MSE and R values of 2.61e-6 and 0.974, respectively. Bayesian Regularisation on the other hand takes more time but can produce good, generalised models for noisy data. This algorithm produced even better statistics of 2.51e-6 and 0.976 on during a random training example. Scaled Conjugate Gradient is similar to Levenberg-Marquardt but sacrifices speed for minimal memory usage and produced values of 4.72e-6 and 0.946 which fail to even come close to competing with its predecessors. The superior training algorithm for the provided problem is therefore Bayesian Regularisation.

As an added exercise, I tested and evaluated the effect of changing the validation and testing data percentages. 80/10/10 produced statistical values of 2.72e-6 and 0.974 while 60/20/20 produced statistical values of 3.26e-6 and 0.967. Neither beat the values produced in the standard 75/15/15 training above so this will remain the preferred training ratio. (Figure 1) shows the final network structure and design, complete with delays, activation functions and biases.

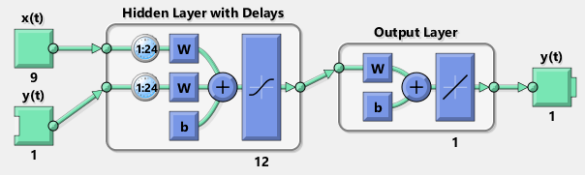

Figure 1: Final Network

Results and Evaluation

Now that we have finalised our model parameters, we can evaluate its strengths and weaknesses through a variety of fitness metrics and plots. We have already heavily discussed the use of mean squared error (MSE) which depicts the average squared difference between expected and actual produced values and the use regression axes which depict how well the target output compares to the neural network produced output. (Figure 2) shows the regression axes at the end of training. Regression during the training set is tight and well regressed as expected, meanwhile the testing set displays minor outliers that reduce the regression R value dramatically yet maintain a respectable R value of 0.969. Over all data sets, we are left with a final R value of 0.991 which implies an extremely strong correlation between output and target values.

We can also study the performance of the network throughout the different epochs in (Figure 3) which shows an initial steep decline in error within the first 10 epochs which then falls slightly less steeply for around the following 20 epochs. As expected, the training set continues to improve at a steady pace until the 100th epoch reaching a best training performance of 5.85e-7. Meanwhile, the testing set plateaus far earlier at around the 25th epoch reaching a performance error of ~e-6.

Another statistic which is useful for studying our model is the error histogram (Figure 4). We can observe a very skinny distribution of errors which could be described as normal centred slightly above the zero-error line. This is a good outcome, however, since we want the histogram to be as skinny as possible such that the magnitude of errors is low.

A key concept behind the recurrent network is the use of sequential time-based data so we also plot a time series response (Figure 5) which describes how the errors change throughout the samples chronologically. An interesting thing to note about this graph is the few spikes in errors present in the summer months, when bikes rented increases. What could have caused these spikes is unknown, but it is likely something within the summer/autumn months since spikes are seemingly absent from winter/spring months. Perhaps errors are more likely here because other outdoor events such as marathons, for example, are more likely to occur in warmer months and would reduce bike traffic dramatically on a singular day.

As a suitable comparison, I also plotted regression axes (Figure 6) and error histograms (Figure 7) for a feedforward network with equal training methods and hidden layers. Clearly shown by its regression graphs, the neural network fails to predict rented bikes effectively with a maximum R value of 0.807. The error histogram is also wider showing that the magnitudes of errors are large.

The only caveat of the produced recurrent network is the long training time which would sometimes take upwards of twenty minutes as opposed to the twenty second training of the feedforward net. Since training only needs to be performed once, however, this is a non-issue. Therefore, we can conclude that the produced recurrent net would be a more than suitable predictor of bike rentals (when provided with environmental weather factors).

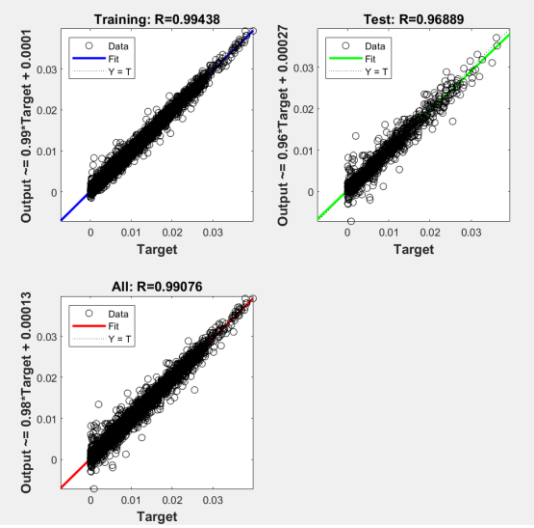

Figure 2: Regression Axes

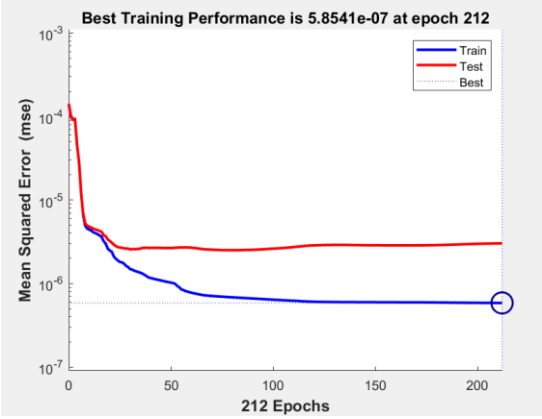

Figure 3: Performance

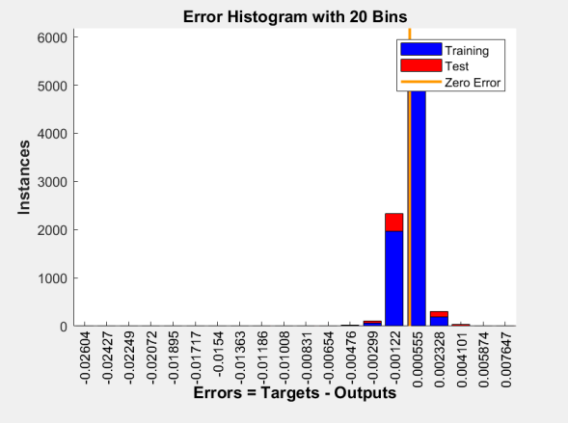

Figure 4: Error Histogram

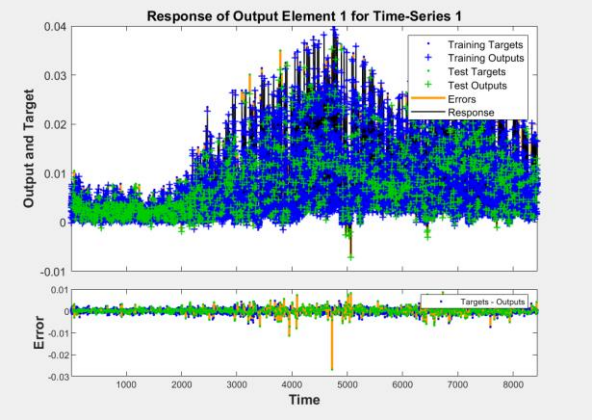

Figure 5: Time Series Response

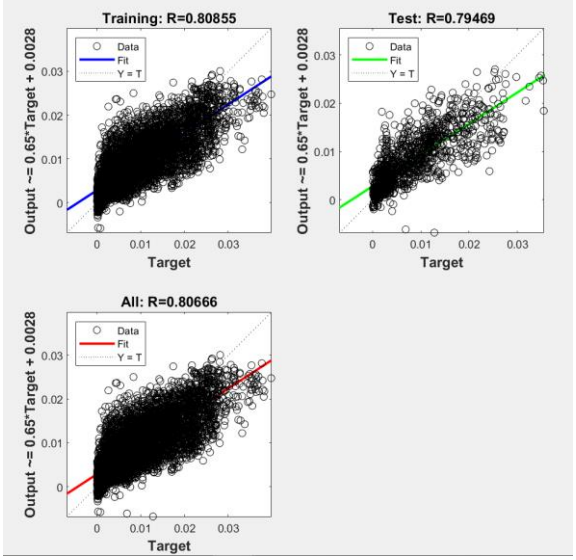

Figure 6: Feedforward Regression

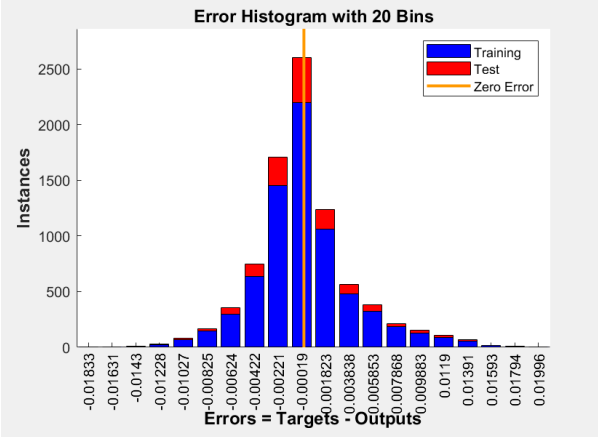

Figure 7: Feedforward Histogram

Further Application

Ethically, it might be seen as immoral to track a user's location data automatically and possibly without their consent. Since the GPS does not tell us the vehicle currently being travelled in,

Like the previous problem, the scenario described and its data are time-based since location is recorded every five seconds and stored sequentially. Therefore, it might also be appropriate to choose a recurrent architecture that includes previous processes as inputs after a provided delay. Since the classification is strongly dependent on the immediate previous process, we would only use a delay of one process. However, if the devices/phones are turned off or lose signal, data wont be retrievable which would break the chain of consecutive time in the data. We would start to see large gaps in the table. Therefore, it might be more appropriate to use a simpler architecture like a feedforward net.

Unlike the previous problem, the scenario describes a classification problem as opposed to a regression function. Therefore, the output layer of the network should contain neurons that correspond to the modes of transport (walk, bike, car, bus etc.) and the input layer will consist of neurons that correspond to each location parameter listed in the table (latitude, longitude, altitude, speed, and acceleration). Multiple techniques can be used here to estimate the class when provided with the output layer values. A simple choice could be made to select the node with the greatest value. Another choice could be pick based on probability, greater nodes have a greater chance of being picked. This is especially good when the network cannot decide between two classes in the layer, so by using chance, the net's decisions are fairer. However, this might also lead to scenarios where miniscule probabilities are chosen by the net through sheer luck despite their lack of confidence.

As for the hidden layer, various numbers of hidden neurons should be trialled and evaluated using appropriate statistics such as mean squared error and by plotting the data's regression. It might also be beneficial to alter the number of hidden layers, but this might lead to overfitting of data where the regression function maps to it's training data too closely leading to poor performance with unseen data.

Once the architecture of the network is finalised, the dataset can be divided between training, validation, and testing sets. The training set is used to fit the model, the validation set is used for fine tuning and the testing set is used to provide an unbiased evaluation of the model. Training the model itself may be complex especially in classification problems so finding an appropriate solution may take a long time. Once a solution is found however, we can extract statistics to describe the model's fitness. This could include mean squared error, which describes the average difference between predicted output neuron values and the expected output neuron values, or by plotting the data's regression a long with calculating the R values. A lower mean squared error is preferable and we would hope to see R values approaching 1. It might also be best to repeat the network' evaluation multiple times since each training process is unique from the random initial weight make-up. Some results may be the result of lucky solutions that produce statistics unrepresentative of their structure.