ECMM416: MSci Individual Project

Investigating The Effect Of Data Complexity On Artificial Neural Network Architecture And Performance

Project Report

Candidate 022579

Abstract

Within pattern recognition, Artificial Neural Networks (ANNs) are very powerful tools used to classify data according to patterns which may or may not be apparent within the data itself. This ability to link pieces of data allows for trained networks to have predictive abilities based upon new data, which can allow for advance warning of potential hazards, such as the appearance of tumorous growths within the body.

However, this power comes at a cost of the design and architecture of the network itself, since the capabilities are directly linked to the amount of processing power (the number of neurons) available. Too few neurons within the network, and it will be unable to find a suitable pattern. Too many neurons, and the network will over-fit the data, effectively memorising the training data and being unable to generalise the pattern. This project explored the effect of the complexity of data on the architecture of the network used to predict it, with a view to minimise training time on real data (which can take additional time to learn) by creating architectures on simplified, simulation data. A link was established between input complexity and architecture, assuming the format of the data remains unchanged, however while the link exists, it does not affect the architecture greatly.

1 Introduction

Within pattern recognition, Artificial Neural Networks (ANNs) are very powerful tools used to classify data according to patterns which may or may not be apparent within the data itself. This ability to link pieces of data allows for trained networks to have predictive abilities based upon new data, which can allow for advance warning of potential hazards, such as the appearance of tumorous growths within the body.

However, this power comes at a cost of the design and architecture of the network itself, since the capabilities are directly linked to the amount of processing power (the number of neurons) available. Too few neurons within the network, and it will be unable to find a suitable pattern. Too many neurons, and the network will over-fit the data, effectively memorising the training data and being unable to generalise the pattern. While an inconvenience within a field such as game playing, it can be devastating when the application is, for example, prediction of urban flooding. Additionally, the amount of possible permutations of architecture make a traditional exhaustive search completely infeasible. Given that training a single network on a single set of training data can take between a few minutes and a few hours, we need to reduce the amount potential architectures very early in the design process, so as to minimise the amount of wasted time and increase the final accuracy of the system.

The aim of this project is to explore the potential for designing a training a network on simplified data sets which capture the general format of the full data set without either the full length or complexity, allowing for quicker network training. We can then explore the potential of using this network design as either the final design, or as a base for expansion for the full data set, removing some of the time required to tune the network design for the final data set. This methodology would also allow network designs to be explored earlier in the development process, or before the full data set has been produced/collected.

To achieve this end, it was decided to experiment with an existing system, the RAPIDS1 urban flood prediction system, which has (at the time of writing) been tested on training data produced in simulation software, but has yet to be tested on real rainfall data. This scenario allows us to both experiment with the potential of finding a link between network design and data complexity, and to produce hard results of RAPIDS1 on real world data.

2 Review

Before undertaking such a project, it is important to understand the potential complexities and problems which can be encountered. Thus, a literature review was undertaken to explore the current state of network design with respect to data complexity and design techniques. Three areas of network design were found to exist, of which two contain a number of technologies. On one side, we have network level design, where a process (usually automated) attempts to design the network in accordance with the presented data, starting with traditionally a very simple network and adding components (such as with NeuroEvolution of Augmenting Technologies[1]). This technique was tested in a number of scenarios, giving good performance in comparison with traditional approaches in problems including an indirectly competitive balancing task and a directly competitive survival task.

While other network level design techniques exist, NEAT showed the most promise in application, and as such was selected to be the potential network level tool. After this selection, the area of neuron level design was explored. While a much smaller area, it produces a very interesting technique referred to as Symbiotic Adaptive Neuro-Evolution[2] (as well as an evolution of the technique, Enforced Sub-Population[3]). These techniques focus on producing neurons which are individually effective at a single part of the problem, then combining them using different network designs which are evolved alongside the neurons. This combination of learning processes produces networks which are both highly effective and adaptable, since the system can quickly change neurons in the design based upon new information, rather than having to relearn the entire pattern. This approach produced several excellent results, most notably with robotic pathfinding scenarios, where the SANE approach not only produced the best performing robot, it also dealt much more effectively with simulated failure on the robot.

The final approach (of consequence) investigated was architectural level design, where the allowed architectures (rather than the architectures themselves) are investigated and evolved. The two mechanisms found, Grammatical Rule design[4] and Block Based design[5], operate on slightly different principals, with the rule design evolving a set of 32 rules, and produced architectures are constructed within the bounds of a random number of these rules. Conversely, the block based design uses a set of pre-constructed network 'blocks', where only the weights need to be tuned, which are then dropped into a grid of fixed size. Both of these techniques fared well on simple problems, however they struggled with more complex applications, and thus were not selected for use.

While both of these techniques selected (NEAT and SANE) produced good results, they require significant overhead with respect to setting up the system and produced results. Given the limited time window of the project, and the overall goal of determining if a link between data sets and network design exists, using one of these techniques would simply be moving the problem around and using time that simply does not exist. Therefore, despite the potential results of these techniques, neither will be used.

3 The RAPIDS1 System

Due to the complexity of the RAPIDS1 system, it is certainly noteworthy to explain its operation. RAPIDS1 is an urban flood prediction model based upon an Artificial Neural Network, and predicts the effect of short term rainfall upon the sewerage system of an urban area in order to predict potential flooding from manholes and other sewerage elements in the near future.

The system itself is trained using a series of training and test storms, previously recorded from real rainfall data or using accurate (but time intensive) simulation software. These storms are recorded in discrete time-steps (2 or 3 minutes) with each one corresponding to the conditions over that time period (for example, rainfall, water level in each sewerage element etc.). These are then collated into a single data file with some additional, non simulation information added (cumulative rainfall and elapsed time, allowing the system to gauge the position in the storm).

When operating, RAPIDS1 takes a storm at a time and begins reading using a time window approach. In this way, the system will only produce an output if the time window is full of valid inputs, and thus will not produce output for the first section of the storm as the window fills. The network takes the full input time window and then produces an output for each manhole individually (each manhole has an output node on the network), before moving onto the next time-step in the storm. This process is repeated throughout the storm, until the time window is not full again, marking the end of the storm.

Before being used for prediction, the system is trained on specific training storms, accounting for approximately 75% of the available training data. The network is evaluated on each storm, before modifications are made using the training function. Once the network is deemed to be fully trained, it is given the remaining 25% of storms to be used as an unseen test, and the full set of results (training and testing) are output to data as an output hydrograph, noting the water level in each sewerage element at every valid time-step.

The actual parameters available to the network architecture are split into two major components, those affecting the input to the network, and those affecting the operation.

3.1 Input Parameters

The input parameters of the network are those that affect the quantity of data submitted to the network for processing. The two of boolean values determine whether to use those data columns as additional inputs (elapsedTime for the time since the storm started, napi for the NAPI, the measure of water already in the ground). The time window size, as the name states, determines the length (in time-steps) of the time window, while the windowAdvance determines how many time-steps ahead the system should attempt to predict.

3.2 Processing Parameters

The processing parameters deal much more with the way the network handles the data given to it. The first parameter is the number of hidden units in the network, all of which are fully connected to input and output. Next is the trainingFunction, which determines which training function the network will use during training (previous testing had found Scaled Conjugate Gradient and Quasi-Newton back propagation to be most effective, so only these two were used). The goal parameter represents the one of the termination criterion available to the training process, lower goals being related to lower error between the training data and training results. Finally is the number of epochs, the other termination criterion in the toolbox, and acts as a maximum amount of training time for the system.

4 Experimental Set-up

In performing this experiment, it was important to use an appropriate set-up to ensure fair and accurate results. In order to attempt to establish a link, two concurrent experiments were run on two different data sets (one a simulation set, one a real data set).

4.1 Experiments

The first experiment tested a series of architectures over the two data sets, using the architecture determined from previous testing as a base. This experiment aimed to comprehensively explore a limited set of network architectures and to be used as a complementary experiment to the second set-up.

The second experiment has a much wider scope than the first, as it aims to explore a much larger quantity of potential architectures. This second experiment uses an evolutionary algorithm to explore far more parameters of the problem, thus increasing the number of potential architectures beyond what can be explored with a comprehensive search. These additional parameters were ones which were fixed during the first experiment, but had not been adequately explored during initial testing to ensure the correct values.

4.2 RAPIDS1 Modifications

Initially, modifications were made to RAPIDS1 in order to facilitate the system running the experiment for prolonged periods without user interaction. This included the capability of RAPIDS1 to take a network architecture as input, and to produce a concise results file as output. Additionally, for the second experiment, an additional MATLAB function and evolutionary algorithm were constructed, the function to interface with the evolutionary algorithm, which was built in Java.

To measure accuracy, a simple gauge was utilised. The predicted value for a given time-step was compared against the actual data value, and if the prediction was within 5% of the actual value, it was considered a pass. This was repeated for all time-steps within the storm, and the overall accuracy for that storm is the percentage of passing predictions against the number of time-steps. For the entire set of storms, the accuracy was subject to a weighted average, with the (unseen) test storms set to four times the value of a (seen) training storm. This weighted average represented the fitness of the network architecture for the evolutionary algorithm, with at least 80% seen as acceptable, 90% as good, and 95% as excellent. By using this metric, we can avoid the problem of over-fitting on the training data (even with 12 training storms and 4 test storms, the test storms impact the performance metric more). Additionally, since this error margin is fairly small, we can be assured that high performance network architectures are matching both pattern and magnitude against the real storm.

4.3 Evolutionary Algorithm

In order to perform experiment two, it was necessary to build an evolutionary algorithm. It was decided to build this in Java, due to ease of implementation and increased potential for progress monitoring and candidate storage. The algorithm itself took an initially random population and evaluated them, producing children based upon the best two parents out of three randomly selected parents. The algorithm then performed a crossover (using the first parent for boolean values) and mutation at a given rate. The algorithm then terminated if either the maximum generation was achieved, or if the population was determined to have converged (monitored using the standard deviation of the population performance).

The first few runs of the algorithm were used to explore the limited set of parameters identical to those which were comprehensively tested in experiment one. After this, the complexity was gradually increased until the full set of parameters was introduced for manipulation.

4.4 Data Files

Since this project centred around the potential relation between network architectures for simulated and real data, it was important to choose appropriate data files. Since only one real data file was available (related to the Dorchester area of testing), only experiments related to the Dorchester simulation data hold any validity. However, other experiments were done on Portsmouth data as initially it had been expected that the Portsmouth would also be available.

Additionally, for all but one run, only a subset of the data was used. Previous testing had

Chart 1
Simulated data (full set)

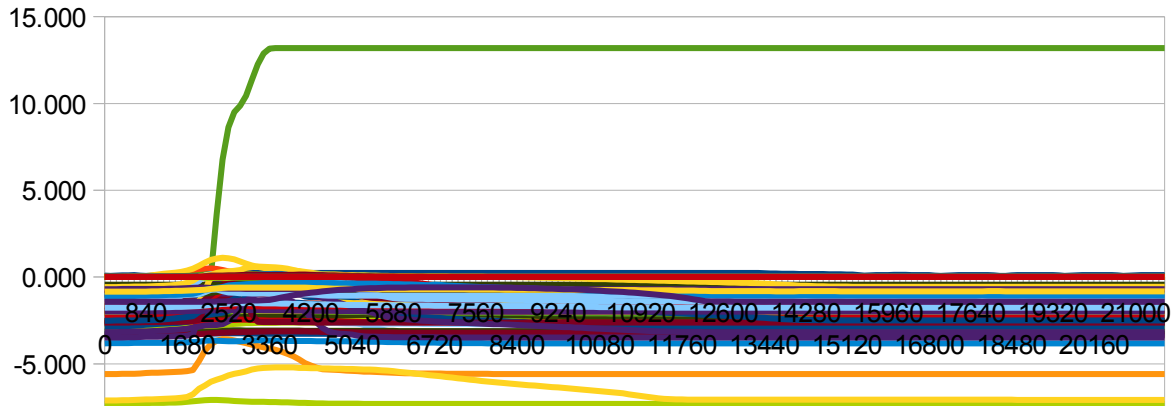
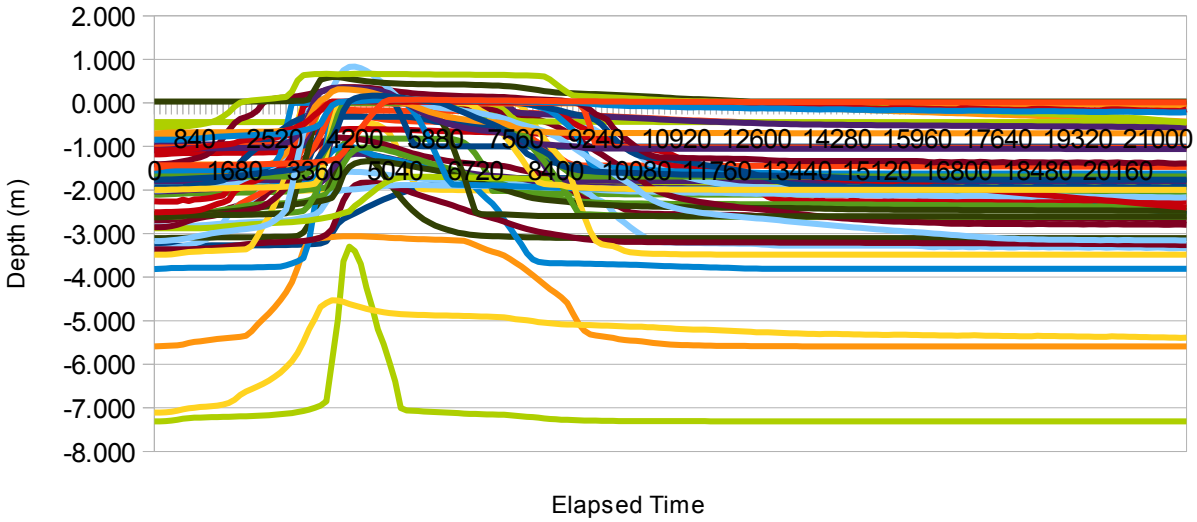


Chart 2
Simulated data (depth only)



shown the network was unable to cope with multiple output signals (i.e. different types of manhole with different units of measurement, see chart 1). As such, the testing was restricted to the manholes using depth (m) measurements (see chart 2). While still hard to read, the second chart only has 2 major shapes for hydrographs, and a smaller range of magnitudes throughout (as opposed to many differing shapes and wildly different magnitudes). As noted, previous testing had shown RAPIDS1 unable to cope with the hydrographs in chart 1, but capable of the subset in chart 2.

Finally, the two sets of rainfall hydrographs (simulated and real) were compared to illustrate the difference in input data complexity against each other. The results of this are shown in charts 3 and 4. The simulated rainfall in chart 3 clearly shows a modified bell curve shape, increasing in magnitude from 0 to maximum intensity, then following the same pattern back down to 0 again. This is in contrast to the rainfalls in chart 4 which, as expected, are much more random, producing multiple peaks and no discernible pattern (due to both the random nature of the rainfall, and the relatively imprecise way of measuring rainfall[6]). Additionally, the storms within the real data last for a much longer time period than the simulation, with simulated storms ending after at most 238 minutes (just under 4 hours), while the real storms continue raining for up to 892 minutes (nearly 15 hours), giving RAPIDS1 much more data to predict.

Chart 3
Simulated data rainfall

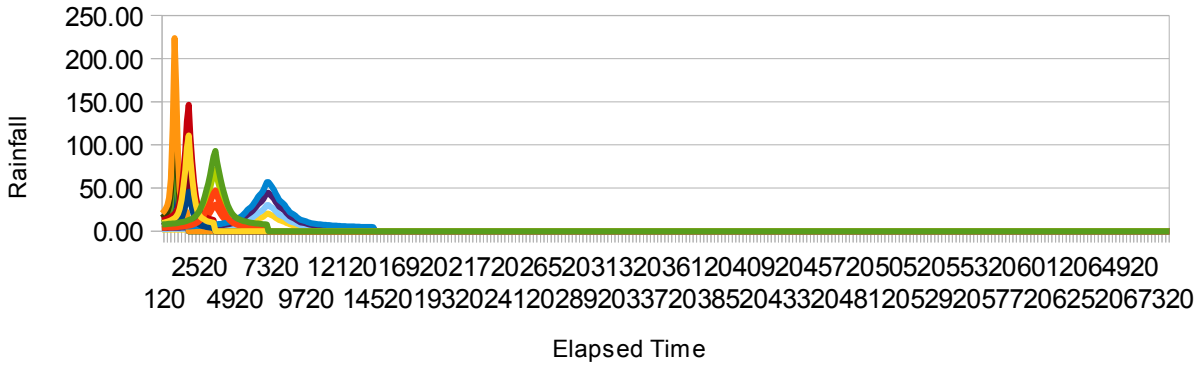
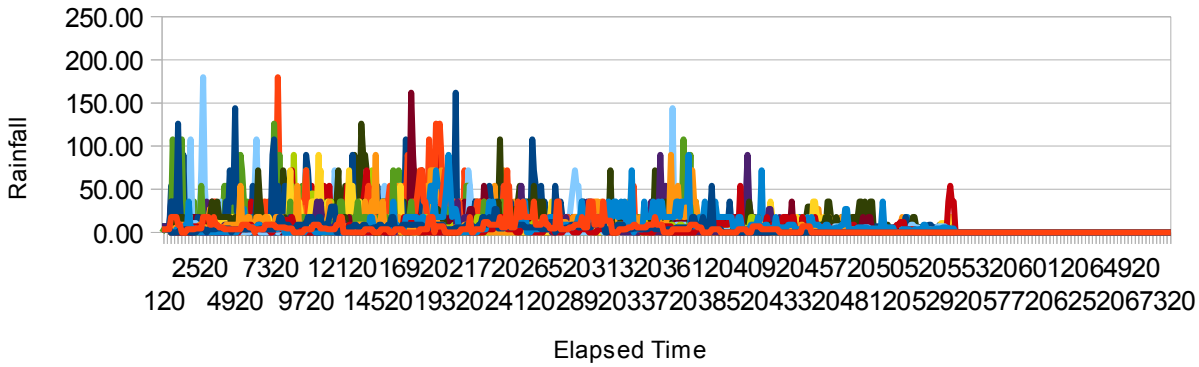


Chart 4
Real data rainfall



5 Methodology

As noted above, these two experiments were carried out in slightly differing ways. The first experiment used it's comprehensive approach as a way of acting as a 'level playing field', exposing both the simulated data and the real data to the same network architectures, and by analysing the results, we can directly compare the effect of data complexity (with respect to input) on the architecture. This allows us to draw exact conclusions on the complexity of the data against the complexity of the architecture, given that each data set was trialled on both sets of architectures. Additionally, it allows us to monitor the peaks of performance with both data sets, again acting as a direct comparison between the two data sets.

The second experiment then allows us to test RAPIDS1 to it's full potential, while still allowing comparison between data sets. The evolutionary algorithm should find optimal (or, at least, near optimal) solutions for the architecture, giving us an indication of the maximum performance available from the system given an ideal architecture. These ideal architectures can then be compared against each other, looking for similarities and differences and thus attempt to find any link between complexity and architecture.

These two experiments have been designed in a complementary manner, allowing the problem to be (effectively) explored from two distinct angles (architecture and performance), while simultaneously allowing for results to be compared. This two stage approach also reduces the chance of failure, as while a single experiment could feasibly fail, the probability of both failing to provide valid results is very unlikely (two failed experiments could be taken as a result in itself).

6 Results

After the experiments had run, the results were collated for analysis. Due to the way the MATLAB Neural Network Toolbox works, it was only necessary to run the tests in experiment one once, since previous testing showed that the toolbox has no (obvious) random element and will reach the same configuration of weights and biases if the same data and network architecture was provided as input.

6.1 Experiment One

As noted in §3.1 the first experiment was performed on a limited subset of all the possible parameters in order to keep execution time within a reasonable frame. The parameters which were explored were the number of hidden units within the network (ranging from 6 to 16) and the goal (ranging from 0.5 to 0.00005), with all other parameters fixed (see appendix A for full list). The results for the simulated data are presented in table 1, while the real data is in table 2.

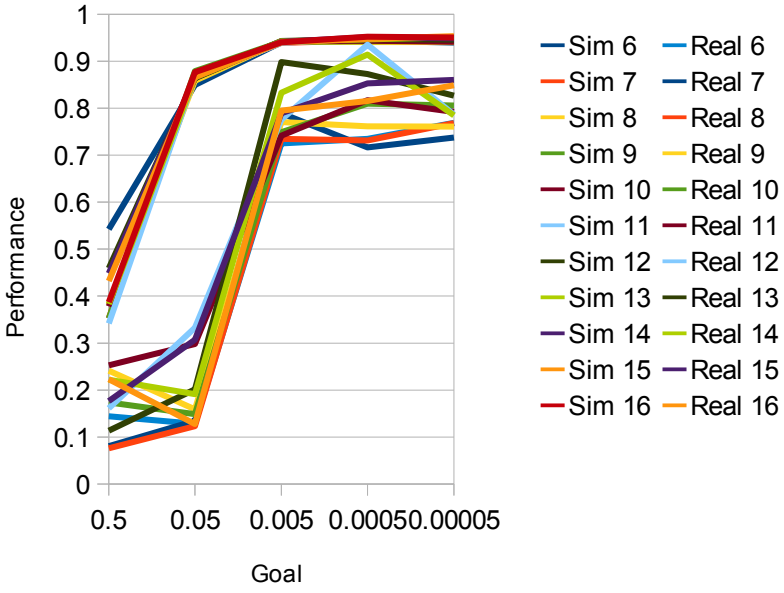
Hidden Unit \ Goal	0.5	0.05	0.005	0.0005	0.00005
6	0.5421	0.8493	0.9401	0.9436	0.9398
7	0.3528	0.8674	0.9389	0.9446	0.9437
8	0.3812	0.8588	0.9422	0.9410	0.9408
9	0.3524	0.8791	0.9413	0.9427	0.9439
10	0.3782	0.8662	0.9423	0.9439	0.9414
11	0.3421	0.8671	0.9394	0.9501	0.9488
12	0.4591	0.8640	0.9435	0.9469	0.9461
13	0.3831	0.8661	0.9423	0.9463	0.9511
14	0.4485	0.8664	0.9412	0.9512	0.9489
15	0.4324	0.8658	0.9400	0.9480	0.9534
16	0.3876	0.8770	0.9406	0.9520	0.9504

Table 1: Experiment 1-I results using simulated data

Hidden Unit \ Goal	0.5	0.05	0.005	0.0005	0.00005
6	0.1443	0.1284	0.7252	0.7344	0.7692
7	0.0805	0.1345	0.7893	0.7167	0.7377
8	0.0761	0.1230	0.7347	0.7311	0.7680
9	0.2414	0.1588	0.7707	0.7612	0.7611
10	0.1738	0.1486	0.7491	0.8101	0.8056
11	0.2526	0.2982	0.7415	0.8173	0.7922
12	0.1611	0.3327	0.7754	0.9352	0.7866
13	0.1138	0.2015	0.8983	0.8729	0.8265
14	0.2220	0.1914	0.8328	0.9141	0.7848
15	0.1765	0.3070	0.7894	0.8528	0.8602
16	0.2230	0.1279	0.7951	0.8152	0.8488

Table 2: Experiment 1-II results using real data

Chart 5
Experiment 1 performance summary



Looking at both tables 1 & 2, and chart 5, it becomes evidently clear that the system struggles to match the pattern if the target goal is too high (i.e. the system has insufficient time to determine the pattern) as shown with the low performance values. However, given sufficient time, the system achieves good accuracy on the simulated data (averaging 94% for goals ≤ 0.005) across all numbers of hidden units, shown by the low standard deviation of 0.4% (goals ≤ 0.005). This appears to demonstrate that on the simulated data, the architecture is, to an extent, irrelevant against giving the system sufficient time to learn the pattern.

In slight contrast, the real data is too complex for small numbers of hidden units, and appears to only reach an acceptable level of accuracy with at least 10 hidden units and a goal ≤ 0.0005. This is attributable to both the longer duration of the storms, and the more random nature of the rainfall within those storms.

To attempt to draw some conclusions from experiment 1, a further set of runs was done, covering the same goal range but with between 1 and 5 hidden units on the simulated data, in an attempt to find the lower limit on the number of hidden units. Table 3 contains the results from these runs, and shows us that even with 1 hidden unit, we get good accuracy,

Hidden Unit \ Goal	0.5	0.05	0.005	0.0005	0.00005
1	0.4358	0.8567	0.9036	0.9030	0.9034
2	0.4447	0.8795	0.9238	0.9237	0.9237
3	0.4071	0.8718	0.9298	0.9299	0.9290
4	0.5034	0.8594	0.9323	0.9367	0.9330
5	0.2875	0.8689	0.9367	0.9347	0.9388

Table 3: Experiment 1-a results using real data

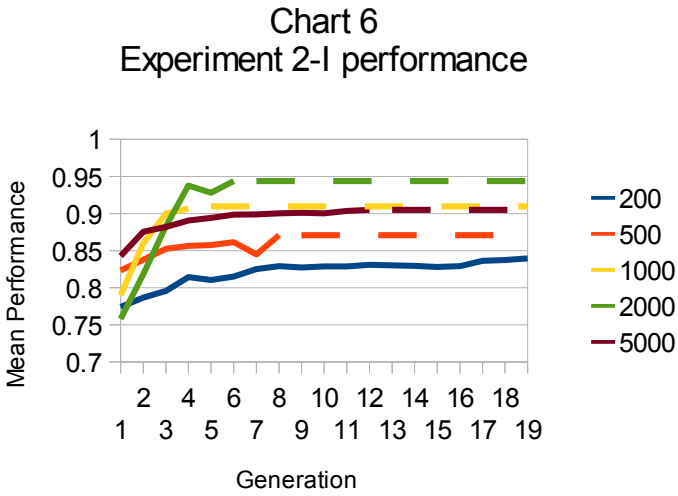
indicating that the data is potentially too simple for RAPIDS1 system to predict, as the necessary predictions can potentially be done without a hidden layer even present. With that said, to achieve the previous levels of accuracy averaged above, we need at least 4 hidden units within the system (and a goal of at most 0.0005).

6.2 Experiment 2

Experiment 2, unlike experiment 1, had a much larger group of parameters to optimise, and due to the limited time available (combined with the quantity of time required to perform a single run). The first set of runs was performed allowing the evolutionary algorithm to optimise every parameter (within certain ranges, see appendix B), and produced the results shown in table 4 and chart 6. Each of the 10 runs (5 epoch limits, 2 repetitions) managed to naturally terminate due to convergence a differing generations. As

Generation	200	500	1000	2000	5000
1	0.7743	0.8233	0.78975	0.7579	0.8426
2	0.7867	0.8378	0.85985	0.8182	0.8755
3	0.7957	0.8523	0.89975	0.8831	0.8817
4	0.8144	0.8565	0.90655	0.9375	0.8904
5	0.8105	0.8576	0.90955	0.92785	0.8939
6	0.8152	0.8613	0.90955	0.94355	0.8984
7	0.8252	0.8451	0.90955	0.94355	0.8987
8	0.8291	0.8709	0.90955	0.94355	0.9004
9	0.8271	0.8709	0.90955	0.94355	0.901
10	0.8286	0.8709	0.90955	0.94355	0.9001
11	0.8287	0.8709	0.90955	0.94355	0.9033
12	0.831	0.8709	0.90955	0.94355	0.9047
13	0.83	0.8709	0.90955	0.94355	0.9047
14	0.8293	0.8709	0.90955	0.94355	0.9047
15	0.8281	0.8709	0.90955	0.94355	0.9047
16	0.8292	0.8709	0.90955	0.94355	0.9047
17	0.8363	0.8709	0.90955	0.94355	0.9047
18	0.8374	0.8709	0.90955	0.94355	0.9047
19	0.8397	0.8709	0.90955	0.94355	0.9047

Table 4: Experiment 2-I results using simulated data



we can see, the limited number of epochs have a noticeable effect on hampering mean performance, as the network is simply unable to reach a low enough goal in the training time allowed, irrespective of network architecture. We can also compare the produced networks architectures, by taking the top performing architectures from each of the different epoch time constraints, as detailed in table 5. These results slow a clear pattern in the architecture, the requirement for comparatively large time windows and low goals (to ensure maximum learning time), combined with an average number of hidden units and little to no advance. NAPI and elapsed time appear to have little effect on the performance, with the network learning from the available data. This is interesting, since the algorithm managed to create a network architecture which could match the performance of those in

Item	200	500	1000	2000	5000
Elapsed Time as input	FALSE	TRUE	FALSE	TRUE	TRUE
Goal	1.07E-005	0.0011557	2.01E-004	0.0016586	1.93E-006
Hidden Units	9	9	9	8	10
NAPI as input	FALSE	FALSE	TRUE	TRUE	TRUE
Time window advance	0	1	0	0	1
Time window size	26	24	22	22	21

Table 5: Produced network architectures from experiment 2-I

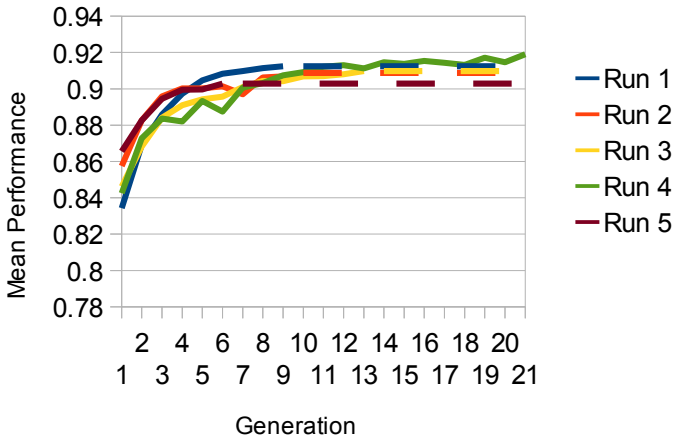
experiment one using just one fifth of the learning epochs.

The second component of the second experiment was the same as the first, with the exception that the evolutionary algorithm was allowed to optimise the number of epochs, ranging from 1 epoch up to 10,000. This was repeated five times, each time reaching convergence criterion. The mean performance of each generation for each of the five runs has been noted in table 6 and plotted on chart 7. As with 1,000, 2,000 and 5,000 epochs in chart 6, we see a steady rise in performance (as we would expect), before tailing off as convergence is reached. Disappointingly, none of the runs managed to reach the 94.4% accuracy achieved by the previous 2,000 epoch run, although the 90-91% is on target with the 1,000 and 5,000 epoch optimisations. As before, the top performing architecture at the end of each run has been tabled along with the parameters selected in table 7. Again, we can see the large time window for input and low advance, as well as low goal. Additionally, NAPI and elapsed time still do not appear to influence the outcome, and we have a large number of epochs (reinforcing the need for extended training to produce the best networks). The only major change from table 5 is the increase in the number of hidden units within the network, although given the results from experiment 1 (where we only witnessed a lower threshold for the number of hidden units), it appears that a number of hidden units above 8 is sufficient with the enlarged time window, and that it is perhaps the small range of numbers from table 6 is coincidental.

Generation	Run 1	Run 2	Run 3	Run 4	Run 5
1	0.8343	0.8575	0.8462	0.8426	0.8656
2	0.8691	0.8826	0.8682	0.8728	0.8826
3	0.8858	0.8958	0.8841	0.8836	0.8946
4	0.8973	0.9002	0.891	0.8821	0.8995
5	0.9047	0.9003	0.8942	0.8933	0.8997
6	0.9083	0.9014	0.8956	0.8875	0.9029
7	0.9098	0.897	0.9009	0.9007	0.9029
8	0.9114	0.9063	0.9043	0.9032	0.9029
9	0.9124	0.9068	0.9042	0.9075	0.9029
10	0.9124	0.9088	0.9068	0.9092	0.9029
11	0.9124	0.9088	0.907	0.9121	0.9029
12	0.9124	0.9088	0.908	0.913	0.9029
13	0.9124	0.9088	0.9098	0.9113	0.9029
14	0.9124	0.9088	0.9098	0.9146	0.9029
15	0.9124	0.9088	0.9098	0.9138	0.9029
16	0.9124	0.9088	0.9098	0.9154	0.9029
17	0.9124	0.9088	0.9098	0.9144	0.9029
18	0.9124	0.9088	0.9098	0.9131	0.9029
19	0.9124	0.9088	0.9098	0.9171	0.9029
20	0.9124	0.9088	0.9098	0.9146	0.9029
21	0.9124	0.9088	0.9098	0.9191	0.9029

Table 6: Experiment 2-II results using simulated data

Chart 7
Experiment 2-II performance



Item	Run 1	Run 2	Run 3	Run 4	Run 5
Elapsed Time as input	TRUE	TRUE	FALSE	FALSE	TRUE
Epochs	7463	8288	8581	8622	9818
Goal	3.39E-004	0.00193113	4.51E-007	5.79E-005	7.17E-003
Hidden Units	18	14	10	14	12
NAPI as input	FALSE	FALSE	TRUE	TRUE	TRUE
Time window advance	0	2	1	0	0
Time window size	21	22	28	28	26

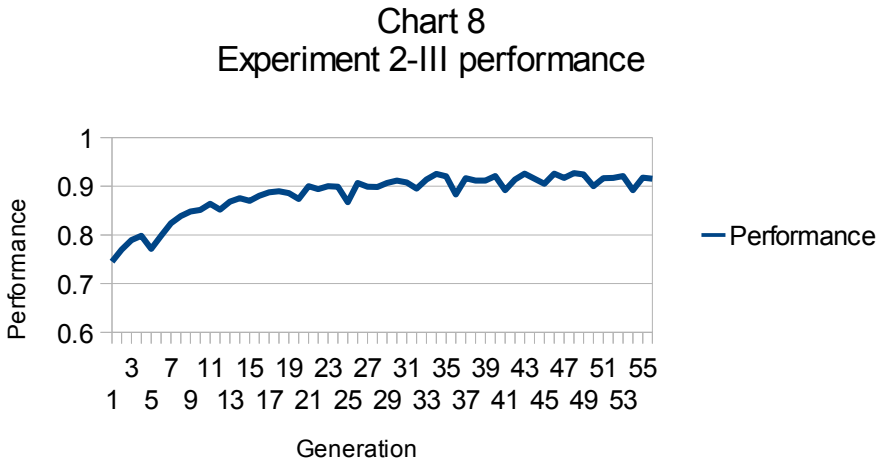
Table 7: Produced network architectures from experiment 2-II

The final part of experiment 2 focussed on the use of the real data, which proved to be much more complex for the system. This is demonstrated by the number of generations achieved (56 as opposed to the previous maximum of 21) and still failed to reach convergence (only achieving 0.04 standard deviation, against the threshold of 0.005), as well as taking considerably longer to execute a generation compared with the previous experiments. As a result, only a single run of this experiment could be made, with the

results presented in table 8 and chart 8.

Generation	Performance
2	0.769649
4	0.79793
6	0.7979955
8	0.8389055
10	0.8510245
12	0.85193
14	0.8751765
16	0.8804695
18	0.889856
20	0.8734545
22	0.894157
24	0.8988745
26	0.9065065
28	0.898189
30	0.911422
32	0.895328
34	0.9250745
36	0.883159
38	0.9116525
40	0.9206645
42	0.9134655
44	0.9156295
46	0.9258335
48	0.926865
50	0.899866
52	0.9167755
54	0.89189
56	0.9151665

Table 8: Experiment 2-III results using real data



Both table 8 and chart 8 show a clear increase in performance from the initial random selection, achieving a final mean of 91.5% (peaking at 92.7% mean at generation 48), in excess of the 80-90% in experiment 1-II. Additionally, the peak performance obtained in experiment 2-III was 95.3% accuracy, again in excess of the 93.5% achieved in experiment 1-II and well in excess of any of the other results achieved in experiment 1-II. However, the area of interest is the architecture generated, and since only one run was completed, the top five performing architectures are noted in table 10. This table holds many similarities to the others produced in experiments 2-I and 2-II, with the large input time window, low goal and no advance. However, we can also note the differences in that both the elapsed time and NAPI inputs appear to have a bearing, as well as the (relatively) low number of epochs compared to experiment 2-II. The real concern from this experiment rests in the single run performed, as the top five performers may have all come from the same parental set, thus the similar architectures. Unfortunately, without additional runs, we can only make rough speculations about the potential impact, and given the time required to run these optimisations (experiment 2-III took nearly 150 hours on a 3.33Ghz Core i5 to complete 56 generations), it was prohibitively expensive to complete multiple runs.

Item	1	2	3	4	5
Elapsed Time as input	TRUE	TRUE	TRUE	TRUE	TRUE
Epochs	6487	6441	6532	6259	6150
Goal	2.72E-004	2.79E-004	2.64E-004	2.77E-004	4.14E-008
Hidden Units	17	17	16	13	14
NAPI as input	TRUE	TRUE	TRUE	TRUE	TRUE
Time window advance	1	0	0	2	0
Time window size	21	21	20	23	20

Table 9: Produced network architectures from experiment 2-III

Even this single run gives us a small chance to compare architectures produced for the two data sets. By looking at tables 7 and 9, we can see a great deal of similarity, both architectures having a similar amount of hidden units and time window size, as well as little/no look ahead and low goals. The only real differences appear in the boolean input flags and the number of learning epochs, although given the limited data set of table 9, this could well be attributed to all five architectures branching from a single parent, particularly given the results from experiment 2-I, which tends to confirm both that more epochs is better, and that the elapsed time/NAPI inputs bear little impact on the final design. This is certainly an interesting comparison, showing that the real data, while more complex, does

not require a more complex network to learn it, thus demonstrating that the correct design of network (theoretically) can be found with a more simplistic representation of the data set.

7 Additional Experiments

In addition to the experiments run above, several other runs were completed while waiting for the real data. However, since the Portsmouth real data was made available later than the Dorchester, and due to the time taken to run the real Dorchester data, the real Portsmouth data could not be run. Although we cannot make a direct comparison between simulated/real data, we can note any similarities in architecture (given the similar format and content of the data) compared against those generated for the Dorchester data.

The format of the experiments is identical to those run before (the two experiments were run concurrently), and thus the first part deals with limited learning epochs but otherwise full optimisation, while the second set of runs allows for completely full optimisation. The first part (experiment 3-I) is presented in table 10 (no chart was produced due to the limited data set), and immediately shows that the evolutionary algorithm took less generations to reach convergence (indeed, one of the 1,000 epoch runs lasted just 1 generation). Additionally, the low number of epochs appears to have had little impact on the performance, with an accuracy gap of just 1.16% between the two extremes of epochs. Given these results and limited number of generations, we could be tempted to conclude that the architecture has little impact, however looking at the highest performing architectures in table 11, we again see the pattern of low goal and high time window, although slightly more variable number of hidden units than table 5, as well as evidence to suggest that NAPI and elapsed time are required inputs, unlike for the Dorchester data. This demonstrates the speed at which the evolutionary algorithm is operating, and thus we can conclude that the results in table 9, despite coming from a single run, are almost certainly valid.

Generation	200	500	1000	2000	5000
1	0.893262	0.794022	0.80926	0.726651	0.773401
2	0.92329	0.891387	0.9045545	0.8206565	0.8243515
3	0.9286845	0.931468	0.937153	0.8991085	0.9297625
4	0.9286845	0.931468	0.937153	0.9410875	0.944805
5	0.9286845	0.931468	0.937153	0.944293	0.944805

Table 10: Experiment 3-I results

Item	200	500	1000	2000	5000
Elapsed Time as input	TRUE	TRUE	TRUE	FALSE	TRUE
Goal	1.75E-003	1.99E-003	9.38E-004	8.52E-008	5.47E-004
Hidden Units	6	6	9	16	12
NAPI as input	TRUE	TRUE	TRUE	TRUE	TRUE
Time window advance	1	0	2	0	1
Time window size	22	24	21	17	24

Table 11: Produced network architectures from experiment 3-I

Experiment 3-II, as with experiment 2-II, then allowed the evolutionary algorithm to optimise the number of epochs given to training. Table 12 provides these results, which are of a similar level to those produced in experiment 3-I. Additionally, the very low range and

Generation	Run 1	Run 2	Run 3	Run 4	Run 5
1	0.600363	0.707905	0.821916	0.7189835	0.740634
2	0.7749485	0.819731	0.8980165	0.859691	0.869844
3	0.8970465	0.893784	0.9328095	0.9270725	0.925871
4	0.9395675	0.9120505	0.9133425	0.933314	0.9031875
5	0.9352725	0.913255	0.907744	0.944156	0.9116115
6	0.9451355	0.9443265	0.9164665	0.944156	0.919661
7	0.9451355	0.9443265	0.916288	0.944156	0.8790415
8	0.9451355	0.9443265	0.9455105	0.944156	0.9452275

Table 12: Experiment 3-II results

standard deviation demonstrate that RAPIDS1 has achieved the maximum level of performance capable on this particular data set. Studying the architectures produced (table 13), we see the same evident pattern of high window size, low goal and little advance. This time, however, we see a much larger range of training epochs, not surprising given the results from experiment 3-I. The variety of hidden units also confirms the earlier conclusion (experiment 1) that the pure number of hidden units does not matter, as the range of hidden units which can be used is quite large for this data set.

Item	Run 1	Run 2	Run 3	Run 4	Run 5
Elapsed Time as input	TRUE	TRUE	TRUE	FALSE	TRUE
Epochs	3328	5201	5599	7334	5263
Goal	6.59E-005	8.79E-004	2.41E-005	2.17E-003	9.89E-005
Hidden Units	19	15	11	17	15
NAPI as input	TRUE	TRUE	TRUE	TRUE	TRUE
Time window advance	1	2	0	2	1
Time window size	19	15	18	19	20

Table 13: Produced network architectures from experiment 3-II

Both of these supplemental experiments serve to reinforce the results of earlier experiments. In particular, experiment 3-II shows us the speed at which the evolutionary algorithm works, which in turn helps validate the results of experiment 2-III. However, the main result from these experiments is the effect of the data, in that while the Portsmouth data is significantly smaller than the Dorchester data in terms of number of outputs, it is capable of operating on the same architecture as the Dorchester data without over fitting the patterns. This effect is most likely due to the fact that the architecture appears to be relying on a large input window, rather than a high number of hidden units, and thus the lower number of hidden units are unable to over-fit.

8 Conclusion

The results of this work have been somewhat surprising, given how much literature reinforces the necessity of a well designed architecture, lest the architecture over or under fits the data presented. In contrast, the results presented here definitely show a lower bound for the number of hidden units, however the upper bound was never evident in testing, as increasing the number of units led to increased performance in the vast majority of cases (and certainly not the performance drop we would expect to see from over-fitting). Given the position of the lower bound on the simulated data in experiment 1, we would have expected to see the upper bound reached, particularly when the number of hidden units reached 4 times the lower bound. Given the amount of evidence to demonstrate the low range of suitable hidden units, we could conclude that the MATLAB Neural Network Toolbox includes some feature to prevent (or reduce) over-fitting, although no such feature is noted in the documentation.

However, the main focus of the investigation was the relation between data complexity and network architecture, and by looking at the experiments conducted, we can see reasonably clearly that input complexity appears to have little impact on the architecture. Within experiment 1, there is a requirement to add a couple of hidden units to the architecture when converting from simulated data to real data. However, given the lack of real upper bound, there would be no requirement to fine tune the increased number of units, as long as sufficient were added (for example, adding 4-6 units would result in reasonable performance, assuming the remaining architecture and configuration was shared).

To an extent, we can confirm this result by looking at experiment 2, where the evolutionary algorithm optimised the architectures. Within the simulated data, we have, on average, 11 hidden units used, while the real data averages 15. Additionally, we managed to achieve very similar performance with the optimised architecture, indicating that the architecture, if correctly configured, can handle data of differing complexity with respect to input. This gives rise to another interesting find from the work presented. Within experiment 2, the results very quickly show that the size of the input window has a much larger bearing that the inputs themselves, a result confirmed within experiment 3. This is a clear indication that large amounts of 1 or 2 variables are preferred over smaller amounts of 3 or 4 variables, at least within this context.

Therefore, overall we can conclude that, from the results available, the architecture, if given a sufficiently large input window, and sufficient time to learn the pattern, is capable of learning the pattern within both the simulated and real data, with minimal change in the architecture. However, given that only a single run of real data was completed, we must approach this conclusion cautiously, and at least complete more runs to confirm this conclusion, particularly using the Portsmouth data (to attempt to establish a more comprehensive pattern).

9 Future Work

From this report, there are one major area of future study which could be conducted. At first glance, it appears we should continue the work detailed here, and attempt to establish a much more concrete relation between input data complexity and network architecture, on a wider variety of systems and data sets. However, given the results described above, and the fact that differing network architectures exist for differing data sets, a more comprehensive study of data complexity against architecture could be undertaken, investigating the relation between the complexity of the output data against the network architecture. As noted in §4.4 the data set was reduced to a single unit of measurement as the architecture of the time was unable to predict against a variety of output units. Since a link between data complexity and architecture must exist, it seems appropriate to investigate the output data next, as the input data complexity appears to have little impact on the architecture of the system. This is somewhat confirmed in the difference in evolutionary time between experiments 2 and 3, where the evolutionary algorithm took less generations to optimise the Portsmouth data, which contains less sewerage elements than the Dorchester data.

10 Acknowledgements

The author would like to thank Andrew Duncan for his work on creating RAPIDS1, as well as his continued support throughout this work.

11 References

[1] – K. O. Stanley, R. Miikkulainen *"Evolving Neural Networks Through Augmenting Topologies"* Evolutionary Computation volume 10, issue 2 (Summer 2002)

[2] – D. E. Moriarty, R. Miikkulainen *"Forming Neural Networks Through Efficient And Adaptive Coevolution"* Evolutionary Computing volume 5, issue 4 (1997)

[3] – F. J. Gomez, R. Miikkulainen *"Solving Non-Markovian Control Tasks With Neuroevolution"* Proceedings of the International Joint Conference on Artificial Intelligence pp1356-1361 (1999)

[4] – X. Yao, Y. Shi *"A Preliminary Study On Designing Artificial Neural Networks Using Co-Evolution"* Proceedings of the IEEE Singapore International Conference on Intelligent Control and Instrumentation pp149-154 (1995)

[5] – S. Moon, S. Kong *"Block-Based Neural Networks"* IEEE Transactions on Neural Networks volume 12, issue 2 (March 2001)

[6] – S.J. Woods, D.A. Jones, R.J. Moore *"Accuracy of Rainfall Measurement for Scales of Hydrological Interest"* Hydrology and Earth Sciences volume 4, issue 4, pp531-543 (2000)

Appendix A: Full parameter listing for Experiment 1

Elapsed Time as input: True

NAPI as input: False

Time window size: 10 time-steps

Time window advance: 0 time-steps

Hidden Units: 6 – 16

Training Function: Scaled Conjugate Gradient

Goal: 0.5 – 0.00005

Epochs: 10,000

Appendix B: Full Parameter listing for Experiment 2-I

Elapsed Time as input: True, False

NAPI as input: True, False

Time window size: 2 – 30

Time window advance: 0 – 10

Hidden Units: 1 – 20

Training Function: Scaled Conjugate Gradient

Goal: 9 – 0.00000001

Epochs: 200, 500, 1,000, 2,000, 5,000