UNIVERSITY OF PRETORIA

Retention Length and Memory Capacity of Recurrent Neural Networks

by

Abraham Daniël Pretorius

Submitted in partial fulfillment of the requirements for the degree
Master of Science (Computer Science)

in the Faculty of
Engineering, Built Environment and Information Technology

December 2020

"Perseverance is the hard work you do after you get tired of doing the hard work you already did."

Newt Gingrich

UNIVERSITY OF PRETORIA

Retention Length and Memory Capacity of Recurrent Neural Networks

Abstract

By

Abraham Daniël Pretorius

Recurrent Neural Networks (RNNs) are variants of Neural Networks that are able to learn temporal relationships between sequences presented to the neural network. RNNs are often employed to learn underlying relationships in time-series and sequential data. This dissertation examines the extent of RNN's memory retention and how it is influenced by different activation functions, network structures and recurrent network types.

To investigate memory retention, three approaches (and variants thereof) are used. First the number of patterns each network is able to retain is measured. Thereafter the length of retention is investigated. Lastly the previous experiments are combined to measure the retention of patterns over time. During each investigation, the effect of using different activation functions and network structures are considered to determine the configurations' effect on memory retention.

The dissertation concludes that memory retention of a network is not necessarily improved when adding more parameters to a network. Activation functions have a large effect on the performance of RNNs when retaining patterns, especially temporal patterns. Deeper network structures have the trade-off of less memory retention per parameter in favour of the ability to model more complex relationships.

Supervisor	: Dr. C.W. Cleghorn
Department	: Department of Computer Science
Degree	: Master of Science

Acknowledgements

This work would not have seen completion without the following people:

- My supervisor, Dr. Christopher Cleghorn for providing invaluable insight, direction and support.
- My parents for support through all the tough times, late night coffees and tricky experiments that kept me up at night.
- The team at Blue Label Telecoms for allowing me to utilise some of their resources and study leave in order to complete the experiments.

Contents

1 Introduction **1**
1.1 Motivation . . . 2
1.2 Objectives . . . 2
1.3 Publication Derived from the Dissertation . . . 3
1.4 Dissertation Outline . . . 3

2 Background: Neural Networks **4**
2.1 Neural Networks . . . 4
2.1.1 Functioning of an Artificial Neuron . . . 5
2.1.2 Feed Forward Neural Network . . . 5
2.1.3 Introduction to Computational Graphs . . . 7
2.1.4 Computational Graph of FFNN . . . 8
2.2 Recurrent Neural Networks . . . 9
2.2.1 Jordan Recurrent Neural Networks . . . 10
2.2.2 Elman Recurrent Neural Networks . . . 12
2.3 Advanced RNNs . . . 13
2.3.1 Long Short-Term Memory . . . 14
2.3.2 Gated Recurrent Units . . . 17
2.4 Bidirectional RNNs . . . 19
2.5 Summary . . . 21

3 Memory Retention of RNNs **23**
3.1 Capacity of Neural Networks . . . 23
3.2 Retention metrics . . . 24
3.3 Memory Retention of Jordan and Elman RNN's . . . 24
3.4 Memory Retention of LSTMs . . . 24
3.5 Memory Retention of GRUs . . . 26

4 Training Algorithms **27**
4.1 Gradient Descent . . . 27
4.1.1 Algorithms . . . 27
4.1.1.1 Batch Gradient Descent . . . 27
4.1.1.2 Stochastic Gradient Descent . . . 28
4.1.1.3 Mini-batch Gradient Descent . . . 29
4.1.2 Momentum . . . 29
4.2 Nesterov Accelerated Gradient . . . 30

4.3 RPROP . . . 30
4.3.1 RMSPROP . . . 31
4.4 Adam . . . 32
4.5 Back Propagation Through Time . . . 33

5 Experiment Performance Metrics 35
5.1 Mean Squared Error . . . 35
5.2 Measuring Memory Retention Performance . . . 36
5.3 Structuring Regression as a Classification task . . . 37

6 Investigation of the Retention Length and Memory Capacity of Recurrent Neural Networks 39
6.1 Experiment Design . . . 39
6.1.1 Constraints Ensuring Fairness of Comparison . . . 40
6.1.2 Spearman Rank Correlation Coefficient . . . 40
6.1.3 Activation Functions to be tested . . . 41
6.2 Experiments . . . 42
6.2.1 Experiment 1: Length of Retention . . . 43
6.2.1.1 Version 1: Regression . . . 43
6.2.1.2 Version 2: One-hot encoding classification . . . 44
6.2.2 Experiment 2: Number of Patterns Retained . . . 44
6.2.2.1 Version 1: Regression . . . 44
6.2.2.2 Version 2: One-hot encoding classification . . . 45
6.2.3 Experiment 3: Frame of Retention . . . 46
6.2.4 Format of each Experiment . . . 47
6.2.5 Implementation Details . . . 48

7 Results 49
7.1 Experiment 1: Length of Retention . . . 49
7.1.1 Average Per Parameter Sequence Length Retention for each RNN 50
7.1.2 Average Per Parameter Sequence Length Retention for each Activation Function . . . 61
7.1.3 Relationship between Sequence Length Retention and the Number Of Layers . . . 77
7.1.4 Relationship between Sequence Length Retention and Different Training Algorithms . . . 88
7.2 Experiment 2: Number of patterns retention . . . 95
7.2.1 Average Per Parameter Capacity Retention for each RNN . . . 95
7.2.2 Average Per Parameter Capacity Retention for each Activation Function . . . 107
7.2.3 Relationship between the Number of Patterns and Number Of Layers 121
7.2.4 Relationship between the Number of Patterns and Different Training Algorithms . . . 133
7.3 Experiment 3: Frame of Retention . . . 143
7.3.1 Average Per Parameter Capacity Retention for each RNN . . . 143
7.3.2 Average Per Parameter Capacity Retention for each Activation Function . . . 152

7.3.3 Relationship between the Number of Frames and the Number Of Layers . . . 161
7.3.4 Relationship between the Number of Frames and Different Training Algorithms . . . 170

8 Conclusion **177**
8.1 Summary . . . 177
8.2 Conclusions from experiments . . . 178
8.3 Future work . . . 180

Bibliography **181**

Chapter 1

Introduction

Neural networks (NNs) have been described as universal approximators by Hornik et al. [28]. From a theoretical standpoint, NNs have no constraints on what can be modelled when the appropriate input, weights and architecture is defined. However, this theorem does not cater for the complexity of learning the parameters or architecture required. Failings of NNs are described by Hornik et al. [28] as a failing in learning, incorrect architecture or a lack of relationship between the input and output mappings. However, multi-layer perceptions do not have the ability to remember information from previous patterns which makes it ill-suited for time series (temporal) predictions.

Recurrent Neural Networks were first introduced in 1986 by Rumelhart et al. [42] to allow NNs knowledge retention of previous observed patterns. Since the RNN can infer knowledge from past observations, the reasoning was that the recurrent connections would be able to learn patterns in temporal (sequential) data where current observations are reliant on previously observed data points. Caveats of simple RNNs were soon discovered. Training RNNs using gradient based optimization methods resulted in either the gradient vanishing or the network weights to grow disproportionately large which had an adverse effect on the effectiveness of these networks [26].

To address these shortcomings, the Long-Short Term Memory (LSTMs) unit was introduced by Hochreiter and Schmidhuber in 1997 [26]. LSTMs remove the vanishing gradient problem of back propagation through time, by introducing gates which control the flow of internal information within the LSTM unit, also referred to as the cell's state. LSTM's are able to retain state information or forget irrelevant stored information when it is required. However, maintaining and forgetting state information is computationally expensive when compared to other more simplistic RNNs, which may lead to an excessive number of training iterations.

Gated Recurrent Unit (GRU) was introduced as an alternative to LSTMs. GRU's reduce the number of parameters required by LSTMs while still retaining the same capabilities [11][12].

The precise impact of different unit types on the memory capacity of RNNs is not well studied. Thus it is difficult to justify using a specific recurrent unit structure as opposed to another. Thus, this dissertation serves to explore different recurrent units and to compare how effective each parameter in the network is utilised for memory capacity and retention.

1.1 Motivation

RNNs were invented to allow NNs to be effectively used for solving temporal learning problems [11][26][42]. However, it is not well known how far back in time steps these networks are able to retain information or how much can be retained. Reasoning about the effectiveness of RNNs information retention have been more indirectly assessed via overall model performance. Usually architectures are iteratively evolved until the performance of the network provide the satisfactory performance.

Performance of the RNN itself is problem dependent as well. If an RNN has a high performance on a temporal set which only contains short-term dependencies, no conclusions on the retention or memory capacity of longer term dependencies can be inferred.

1.2 Objectives

This dissertation is a study of the duration of memory retention and capacity of RNNs. It is also investigated what effect different activation functions and training algorithms have on RNNs with regards to memory retention and capacity. Additionally, the effect of increasing the input dimension on these metrics is also investigated.

The primary objectives of this dissertation are summarised as follows:

- Determine the number of patterns that can be retained by the RNNs (memory capacity)
- Determine the length of the sequence that can be retained by the RNNs (memory length)
- Test the frame capacity of RNNs. Where frame capacity refers to finding a relationship between the length of the sequences and the number of sequences retained.

1.3 Publication Derived from the Dissertation

From the research done within this dissertation, a single journal article has been produced titled ”In Time Memory Capacity of Recurrent Neural Networks”. The produced journal article has been submitted to Elsevier to be reviewed and possibly published under the Neurocomputing journal [1].

1.4 Dissertation Outline

The rest of the dissertation is structured as follows: First the relevant theories on the RNNs which are considered in this dissertation is discussed in chapter 2. Previous works done on memory retention and capacity within RNNs is discussed in chapter 3. Following the background on neural networks, memory retention and capacity, various training algorithms are discussed in chapter 4. Background information on performance metrics for neural networks are then discussed in chapter 5. Thereafter the experimental setup for the various cases investigated are discussed in chapter 6. The experimental findings and discussion is presented in chapter 7. Finally, the findings of this dissertation are summarised in chapter 8.

[1]The home page of the Neurocomputing journal can be found here `https://www.journals.elsevier.com/neurocomputing`

Chapter 2

Background: Neural Networks

This chapter provides a detailed description of RNNs, starting with NN fundamentals, followed by a discussion of previous work done relating to memory retention. First the concept of Neural Networks is discussed in section 2.1. Building on the NN fundamentals, RNNs are discussed in section 2.2 accompanied by two classic architectures; the Jordan RNN 2.2.1 and Elman RNN in section 2.2.2. More complex RNN Neurons follow in sections 2.3.1 and 2.3.2 with a detailed discussion of the Long-short Term Memory (LSTM) and the Gated Recurrent Units (GRU) respectively.

2.1 Neural Networks

The artificial neural network, more commonly referred to as a neural network (NN) is a machine learning model that learns from labelled data ingested by updating weights, combining these weights and adjusting thresholds for decision boundaries. The NN is inspired by the functioning of neurons and synapses within the brain of biological life such as humans [40]. Within a neural network, neurons are commonly referred to as nodes and synapses as edges in literature [30].

For a neural network to learn, a typical strategy that is followed is back-propagation of an error signal using gradient descent which was first invented by Werbos et al. in 1974 [47] and popularised by Rumbelhart et al. in 1986 [42]. Back propagation of the error adjusts weights on the synapses intelligently as to improve future mappings of input to expected output (details of which is presented in section 4.1.1).

2.1.1 Functioning of an Artificial Neuron

Neurons are simplistic units of computation. Input is received by multiplying the values presented to incoming edges with the weight value associated with each respective edge. The result of the products is summed together and presented to an activation function. The activation function then produces a value that is fed along the output edges of the neuron [30].

$$net_y = (\mathbf{v} \cdot \mathbf{x} + b) \tag{2.1}$$

$$y = \sigma(net_y) \tag{2.2}$$

The functioning of the neuron is presented in equations (2.1) and (2.2) where $\boldsymbol{x}$ is the input vector, $\boldsymbol{v}$ the weight vector, b the bias of a neuron, $\cdot$ element-wise multiplication, σ represents the activation function which is typically non-linear and y is the output [30]. The functioning of a neuron is also visually represented in figure 2.1.

Artificial neurons, as described earlier in this section, are combined together to form neural networks. Neurons are divided into layers based on where in the structure they occur in. These networks contain at least an input layer and an output layer, however it is common practice to use one or more hidden layers. The architecture of a basic feed forward artificial neural network is presented in figure 2.2.

2.1.2 Feed Forward Neural Network

The structure of a typical Feed Forward Neural Network (FNN) with one hidden layer is presented in figure 2.2. In this figure, hidden nodes are represented by h_1 up to h_J. Input nodes by x_1 to x_I and output nodes as y_1 to y_K. Weights connecting a neuron i in the

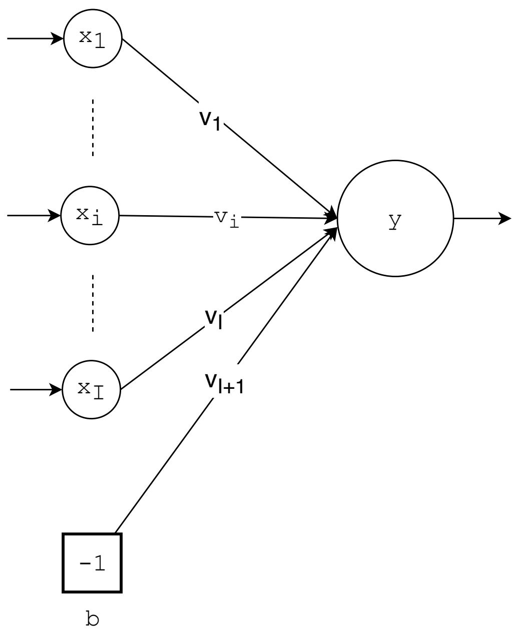

FIGURE 2.1: The internal workings of an artificial neuron [33]

input layer to a neuron j in the hidden layer represented as v_{ji}. A connection connecting neuron j in the hidden layer to neuron k in the output layer is then represented by w_{kj}.

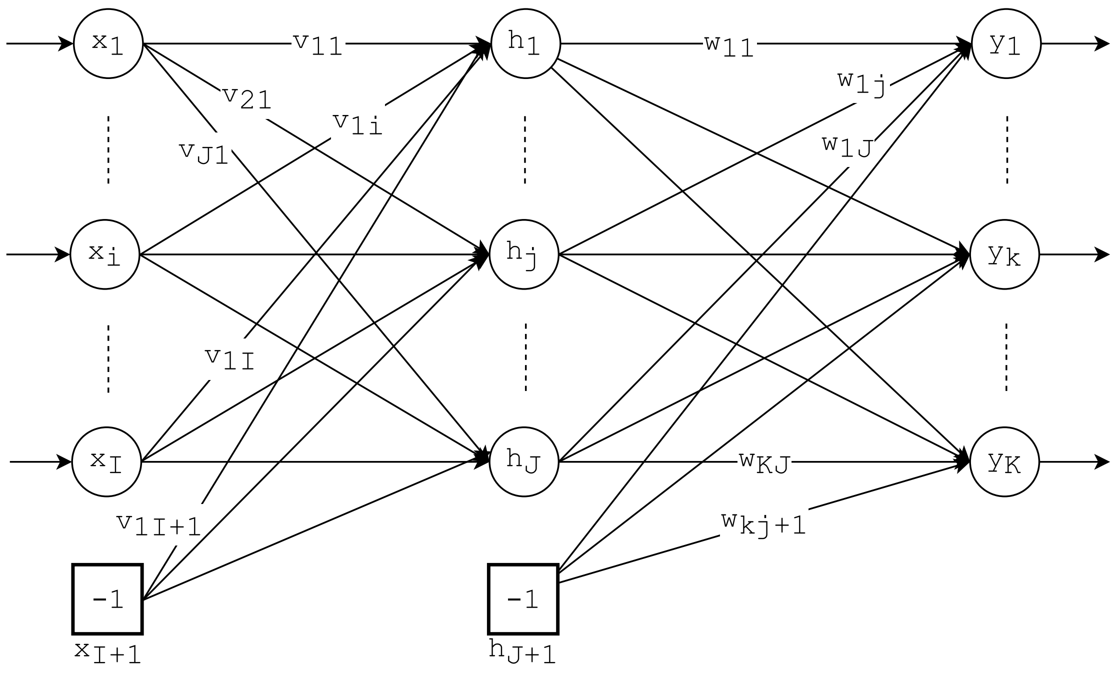

FIGURE 2.2: The architecture of a basic artificial neural network [18]

A FFNN is the connection of many neurons in layers. A typical FFNN consists only of three layers namely: input, hidden and output layers. However, many hidden layers can be used. The resulting activation values from one layer is passed to the next, applying a weight to each neuron's activation value. Thus feeding the information from the input layer across the network to output layer. To calculate the output of a neuron y_k for a single pass through the network using input x and the output vector h of the hidden layer; the equation (2.3) is used. Equations (2.3) utilises equation (2.1). Here σ_{y_k} is the activation of neuron y_k and σ_{h_j} the activation of hidden neuron h_j.

$$\begin{aligned} y_k &= \sigma_{y_k}\Big(net_{y_k}\Big) && (2.3)\\ &= \sigma_{y_k}\Big(\sum_{j=1}^{J+1} w_{kj}\sigma_{h_j}(net_{h_j})\Big) && (2.4)\\ &= \sigma_{y_k}\Big(\sum_{j=1}^{J+1} w_{kj}\sigma_{h_j}(\sum_{i=1}^{I+1} v_{ji}x_i)\Big) && (2.5)\end{aligned}$$

2.1.3 Introduction to Computational Graphs

Computational graphs are used to indicate the flow of a set of operations. Computational graphs are well suited to conceptualise the flow of operations of neural networks, especially the more complex these neural networks become. The representation discussed in this section follows the notation and standards used by Theano [6] and Tensorflow [2]. The representation standard is also followed by Goodfellow et al. in their landmark book *Deep Learning* [22].

A node in computational graphs represent a variable which can be scalar, vector, matrix, tensor, or any other appropriate representation of a variable. Directed edges between nodes represent operations. Multiple edges flowing into the same node from other nodes represent a single operation. Operations only produce one output variable which would then be represented as a node. Intermediate expressions, which are not named in the algebraic representation of the operation, are named in the graph. To highlight these expressions, they are expressed as $u^{(i)}$, indicating the i-th intermediate step.

An example of a computational graph for the functioning of a single neuron is given in figure 2.3.

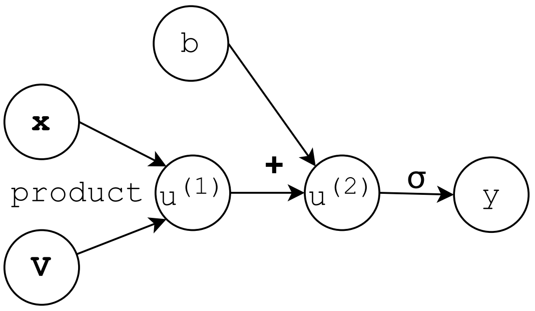

FIGURE 2.3: Computational Graph of a Neuron

In figure 2.3, $\mathbf{x}$ is the input vector and $\mathbf{V}$ is the weight matrix which are then combined through a product operation to produce the intermediate step value $u^{(1)}$. A bias b is added to $u^{(1)}$ which produces $u^{(2)}$. Thereafter an activation function σ is applied resulting in the output value y.

A more condensed form of the computational graph language is then used to indicate flow of operations at a higher level to keep the graph from becoming convoluted [22]. Instead of representing the weight matrices as nodes, the edge represents an operation on $\mathbf{x_t}$ that is parametrised by the weight matrix $\mathbf{V}$. The computation on the edges implies that the activation function will also be applied to the resulting set. The bias variable also becomes a part of the input variable and is not represented explicitly. The condensed representation of a single neuron would then be represented as in figure 2.4.

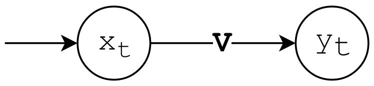

FIGURE 2.4: **Condensed Computational Graph of a Neuron**

2.1.4 Computational Graph of FFNN

A neural network can also be represented using a computational graph as in figure 2.5. The computational graph of a FFNN is simplistic, however it is useful for expanding intuition of state transition between the presentation of patterns to the network when comparing against other architectures such as the recurrent neural networks.

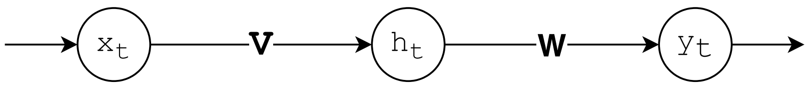

FIGURE 2.5: **Condensed Computational Graph of a FFNN**

2.2 Recurrent Neural Networks

Recurrent neural networks were first introduced by Rumelhart et al. [42], and designed to keep track of the internal state of the network within the context layer. The premise is that, since RNN's keep track of their internal state, these networks would work well with temporal data. The computational graph containing the unrolled operations of the RNN is presented in figure 2.6, which illustrates the feedback of information over time. Unrolling operations is the process of presenting the output $\boldsymbol{o_{t-1}}$ and the activation vector $\boldsymbol{h_{t-1}}$ of the RNN at the previous time step $t-1$ as inputs to the RNN at the current time step t alongside the current input vector $\boldsymbol{x_t}$. At time step t, $\boldsymbol{h_t}$ is the current activation vector of the hidden layer, $\boldsymbol{V}$ is the weight matrix between input and hidden layers. $\boldsymbol{U}$ is the weight matrix between hidden state transitions and $\boldsymbol{W}$ is the weight matrix between the hidden and output layer.

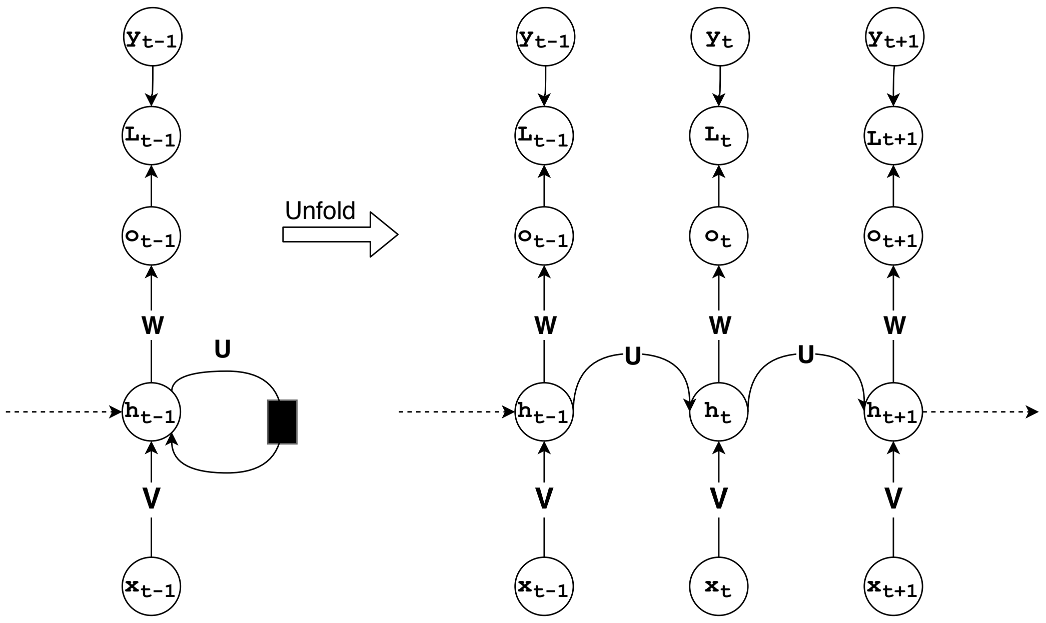

FIGURE 2.6: **Computational graph of a RNN**

At time t the current input vector $\boldsymbol{x_t}$ is fed into the RNN. The previous hidden activation vector $\boldsymbol{h_{t-1}}$ is fed back into the hidden layer. The hidden layer first computes the weighting of $\boldsymbol{x_t}$ by multiplying the vector with the weight matrix $\boldsymbol{V}$, then the same is done between $\boldsymbol{h_{t-1}}$ and $\boldsymbol{U}$. The resulting vectors are then combined and activation functions $\boldsymbol{\sigma_h}$ in each hidden node are applied to produce the new hidden state $\boldsymbol{h_t}$. Since the previous state is also used to produce the new activation, the network is able to learn from previous information. The $\boldsymbol{h_t}$ vector is then fed into the output layer, weighted by $\boldsymbol{W}$. The output layer then produces vector $\boldsymbol{o_t}$ by applying activation functions $\boldsymbol{\sigma_o}$ for each node. Once the output of the network has been obtained, it is compared to the desired vector $\boldsymbol{y_t}$ using the computed loss $\boldsymbol{L_t}$ determined by some loss function.

The equations for the computational graph shown in figure 2.6 are presented in equations (2.6) and (2.7). Bias constants b_h and b_y, which are constant vectors usually set to 1 or -1, are also included in the calculation of the activation values in the hidden and output layer respectively.

$$\boldsymbol{h_t} = \boldsymbol{\sigma_h}\big(\boldsymbol{V x_t} + \boldsymbol{U h_{t-1}} + b_h\big) \tag{2.6}$$

$$\boldsymbol{y_t} = \boldsymbol{\sigma_y}\big(\boldsymbol{W h_t} + b_y\big) \tag{2.7}$$

Expanding on this concept, two general types of simple RNNs can be formed, namely the Elman and Jordan RNNs. Both simple RNNs follow the same state transition as described in this section, however which layer's state is transferred back differs.

2.2.1 Jordan Recurrent Neural Networks

The Jordan Recurrent Neural network was introduced by Jordan in 1986 [32]. In the introductory work a three layer NN was presented which feeds the previous time step's activation vector weighted back into the hidden layer. The previous output of the network is stored in a *state layer*[32]. Jordan reasoned that by explicitly representing the temporal data of the previous time step, the network would be capable of determining relationships between patterns at different time steps [32]. Plan vectors (weights between the hidden and state layer) would then contain, encoded within them, underlying relationships in the temporal data. These weights are updated as the network trains as they should continuously be updated over time [32]. The architecture of the Jordan RNN is presented in figure 2.7. In this figure the previous output stored in $\mathbf{z}$ and the effect thereof on the current hidden state, is controlled by the weight matrix $\boldsymbol{U}$.

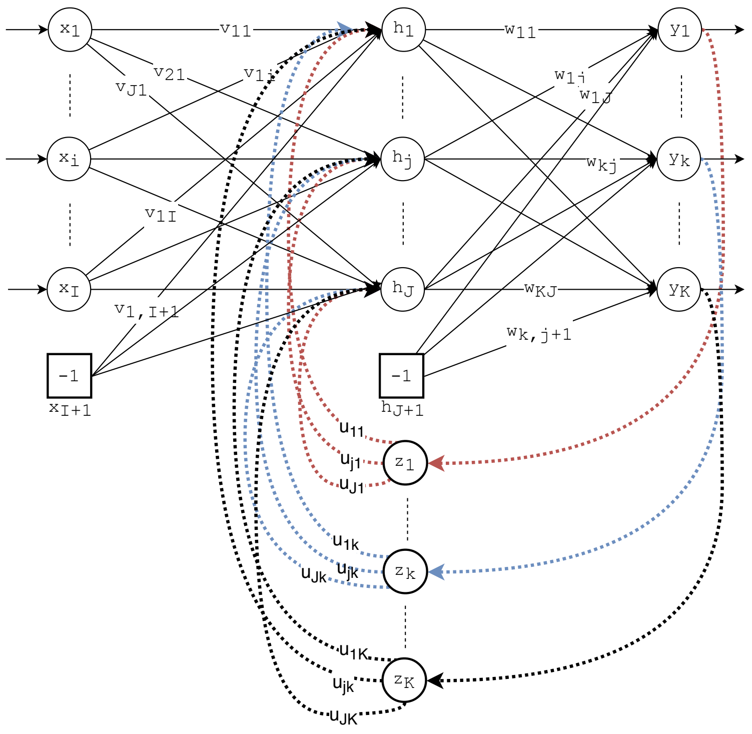

FIGURE 2.7: **Architecture of the Jordan RNNs**

The state transition for the Jordan network is calculated using equations (2.8), (2.9) and (2.10), as defined in section 2.2. Here $\boldsymbol{z}$ is the output vector of the network at time step $t-1$, and the value of $\mathbf{z}$ is referred to as the *state space*.

$$\boldsymbol{z} = \boldsymbol{y_{t-1}} \tag{2.8}$$

$$\boldsymbol{h_t} = \boldsymbol{\sigma_h}\big(\boldsymbol{V}\boldsymbol{x_t} + \boldsymbol{U}\boldsymbol{z} + \boldsymbol{b_h}\big) \tag{2.9}$$

$$\boldsymbol{y_t} = \boldsymbol{\sigma_y}\big(\boldsymbol{W}\boldsymbol{h_t} + \boldsymbol{b_y}\big) \tag{2.10}$$

Building upon these equations, the computational graph of the Jordan RNN is presented in figure 2.8

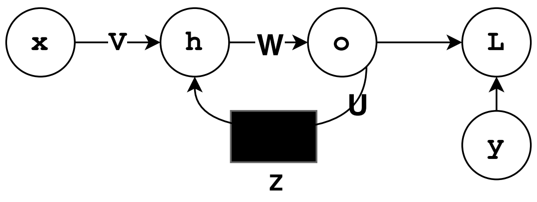

FIGURE 2.8: **Computational Graph of the Jordan RNNs**

2.2.2 Elman Recurrent Neural Networks

Elman RNN was introduced in 1990 by Elman [17]. Elman RNN's store the previous hidden state in a *context layer* instead of the previous output vector in the *state layer* of the network as in the case of the Jordan RNN [17]. Elman's reasoning for this is that by using the previous output vector, as is the case with a Jordan RNN, the network loses the ability to discriminate between relative and absolute temporal information.

Figure 2.9 presents the architecture of the Elman RNN. Here the $\boldsymbol{U}$ represents the weight matrix between the context and hidden layer. This weight matrix controls the effects of the previous hidden state on the current hidden state.

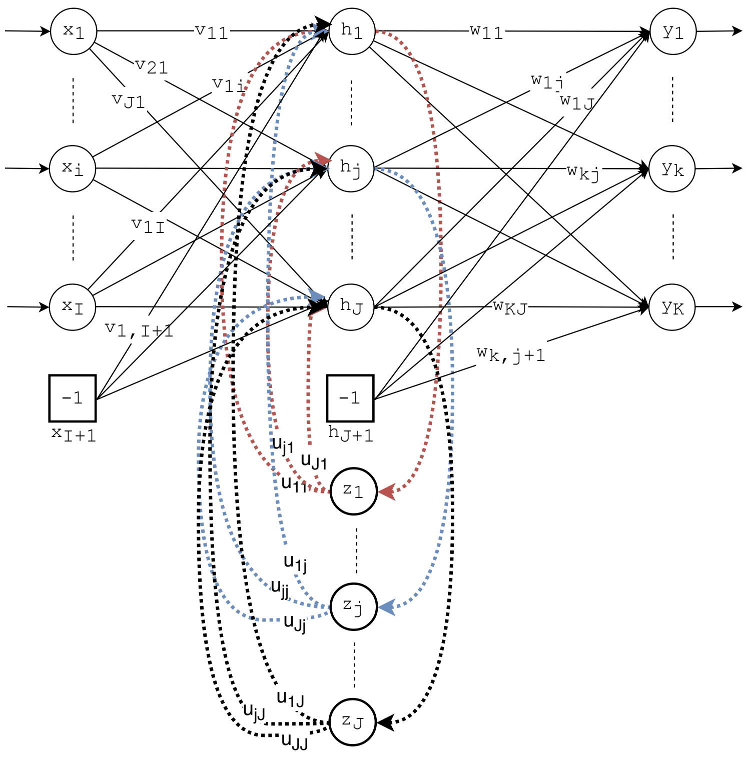

FIGURE 2.9: **Architecture of the Elman RNNs**

The Elman RNN is fully defined by equations (2.11), (2.12) and (2.13).

$$\boldsymbol{z} = \boldsymbol{h_{t-1}} \tag{2.11}$$

$$\boldsymbol{h_t} = \boldsymbol{\sigma_h}\big(\boldsymbol{V x_t} + \boldsymbol{U z} + \boldsymbol{b_h}\big) \tag{2.12}$$

$$\boldsymbol{y_t} = \boldsymbol{\sigma_y}\big(\boldsymbol{W h_t} + \boldsymbol{b_y}\big) \tag{2.13}$$

In equations (2.11), (2.12) and (2.13); $\boldsymbol{z}$ represents the output vector of the hidden layer at time step $t-1$, also known as the *context layer*[17]. This contains the previous hidden state of the current layer into which it is fed, with the effects thereof controlled by the weight matrix $\mathbf{U}$.

Building upon these equations, the computational graph of the Elman RNN is presented in figure 2.10

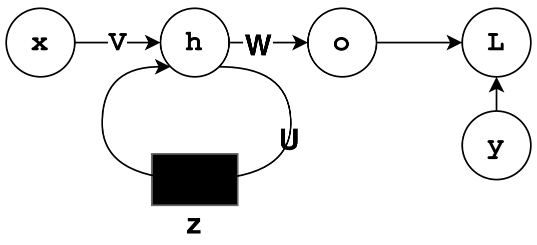

Figure 2.10: **Computational Graph of the Elman RNNs**

2.3 Advanced RNNs

Simple RNNs, such at the Elman and Jordan RNN, have a critical flaw called the vanishing gradient problem [5][25]. Vanishing/exploding gradients was first discovered in deep neural networks. As these networks have many layers and weights, the gradient of the error observed in the output layer becomes more obfuscated as it is backwards propagated through to train the NN [5][25]. This results in either the gradient having no effect on the weights of layers further from the output layer, or that these weights would change too much. The continued use of the chain rule in obtaining the gradient of the error, with respect to a weight far back in the model, leads to products that either collapse to near zero to become increasingly large as depth is increased. This phenomenon typically happens as RNNs are unfolded, essentially becoming deep FFNN's, leading to RNNs being unable to capture long term dependencies [5][25]. To address the issue of the gradient vanishing, more advanced RNN neuron structures were developed, as will be discussed next in sections 2.3.1 and 2.3.2.

2.3.1 Long Short-Term Memory

Long Short-Term Memory is a type of recurrent neural unit introduced by Hochreiter and Schmidhuber in 1997 [26]. An improvement on this structure was presented in 1999 with the inclusion of the forget gate by Gers et al. [21]. This addition was such a large contribution that research regards this inclusion as the basic LSTM. The forget gate addressed a fundamental flaw of the LSTM; As sequences became longer, the network would become unstable as the information learned would interfere with the expected outcome. However, the original LSTM RNNs performed considerably better than the classic RNNs [26]. With the inclusion of the forget gate, the LSTM RNNs become more stable as the LSTM gained the ability to reset its own memory when new sequences were observed [21].

The architecture of a LSTM neuron is presented in figure 2.11. The equations explained within this section will heavily refer back to this figure. It is important to note that this is only a single node of the network and not a whole network on it's own. In figure 2.11 nodes contained within double circles represent state/context nodes which stores the state of the network and passes it on in the next iteration. Here $\mathbf{x_t}$ is the input vector at time step t. Activation functions are represented by σ, τ and μ. The output produced from these activation functions at the current time step is represented by $\mathbf{f_t}$, $\mathbf{i_t}$ and $\mathbf{O_t}$ respectively. Nodes marked by $\times$ are the dot product of vectors flowing in. Nodes marked with $+$ represent the vectors sum of vectors flowing in. The context vector of the neuron at the current time step is $\mathbf{C_t}$ and $\mathbf{C_{t-1}}$ is at the previous time step. Similarly, the output vector of the LSTM neuron at the current time step is $\mathbf{h_t}$. Thus $\mathbf{h_{t-1}}$ represents the previous output vector at time step $t-1$.

Initially the input vector $\mathbf{x_t}$ and the previous output vector $\mathbf{h_{t-1}}$ are added after each has been multiplied with a weight matrix. Then an activation function σ is applied to generate a factor $\mathbf{f_t}$ as in equation (2.14). This step is what is called the *forget gate* [21]. Here $\mathbf{W_{fx}}$ and $\mathbf{W_{fh-1}}$ represents the weight matrices of the forget gate and $\mathbf{b_f}$ is the bias vector.

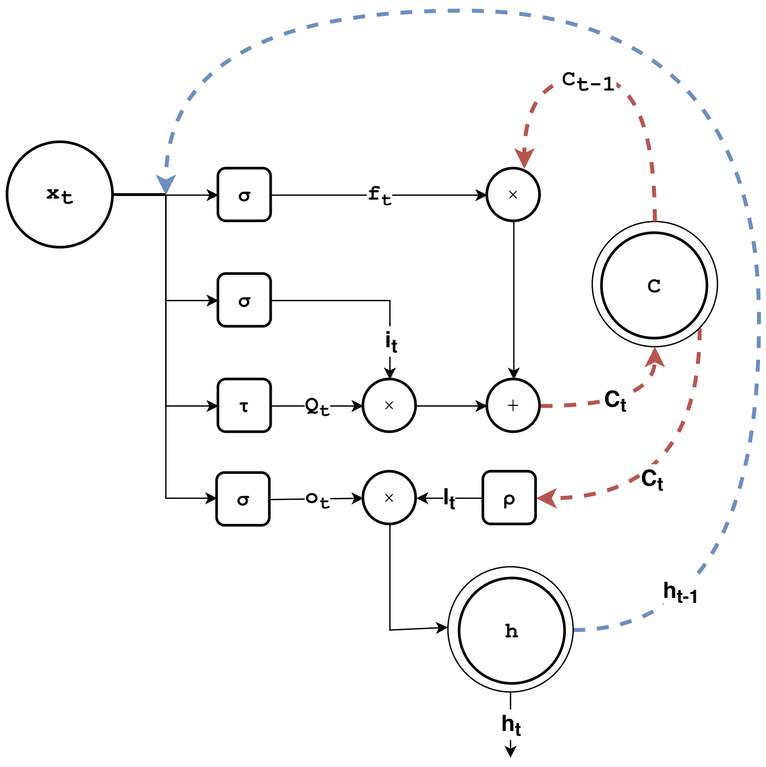

FIGURE 2.11: **Architecture of the LSTM Neuron**

$$\mathbf{f_t} = \sigma\big(\mathbf{W_{fx}x_t} + \mathbf{W_{fh-1}h_{t-1}} + \mathbf{b_f}\big) \tag{2.14}$$

The next step is to determine factor $\mathbf{i_t}$, which is the output of the *input gate* as in equation (2.15)[21]. Here $\mathbf{W_{ix}}$ and $\mathbf{W_{ih-1}}$ are the input weight matrices and $\mathbf{b_i}$ the bias vector.

$$\mathbf{i_t} = \sigma\big(\mathbf{W_{ix}x_t} + \mathbf{W_{ih-1}h_{t-1}} + \mathbf{b_i}\big) \tag{2.15}$$

The intermediate context factor $\mathbf{Q_t}$ is calculated as in equation (2.16).

$$\mathbf{Q_t} = \tau\big(\mathbf{W_{qx}x_t} + \mathbf{W_{qh-1}h_{t-1}} + \mathbf{b_q}\big) \tag{2.16}$$

Combining the input factor and the context factor determines how much of the current information context state will be kept and fed back into the network. The old state $\mathbf{C_{t-1}}$ is then multiplied by $\mathbf{f_t}$ to determine how much information from the past to forget. These two calculations are combined to form the new state $\mathbf{C_t}$ for the current time step as in equation (2.17).

$$\mathbf{C_t} = \mathbf{f_t} \circ \mathbf{C_{t-1}} + \mathbf{i_t} \circ \mathbf{Q_t} \tag{2.17}$$

In the last phase the output of the neuron $\mathbf{h_t}$ is determined. For the vectors of $\mathbf{h_{t-1}}$ and $\mathbf{x_t}$ are first weighted by matrices $\mathbf{W_{oh_{t-1}}}$ and $\mathbf{W_{ox}}$ respectively. A bias $\mathbf{b_o}$ is then added and an activation function σ applied to produce the output of the *output gate* [21] as in equation (2.18). An activation function ρ is then applied to the current state $\mathbf{C_t}$. Both the results of these operations are multiplied together to produce the output $\mathbf{h_t}$ of the neuron at the current time step as described in equations (2.19) and (2.20).

$$\mathbf{o_t} = \sigma\big(\mathbf{W_{ox}}\mathbf{x_t} + \mathbf{W_{oh-1}}\mathbf{h_{t-1}} + \mathbf{b_o}\big) \tag{2.18}$$

$$\mathbf{l_t} = \rho(\mathbf{C_t}) \tag{2.19}$$

$$\mathbf{h_t} = \mathbf{o_t} \circ \mathbf{l_t} \tag{2.20}$$

The full computational graph of a single LSTM node is presented in figure 2.12.

In figure 2.12 multiple occurrences of $\mathbf{x}$ and $\mathbf{h_{t-1}}$ nodes are presented, however each represents the input vector and the previous step's hidden state respectively. These nodes are used to make the graph less complex. All other symbols represent the steps outlined in equations (2.14), (2.15), (2.16), (2.17), (2.18), (2.19) and (2.20).

Figure 2.13 represents the functioning of the LSTM neural network over time steps and utilizes the equations as discussed for figure 2.12. Here the context state of the LSTM is highlighted to distinguish itself from other RNN variants.

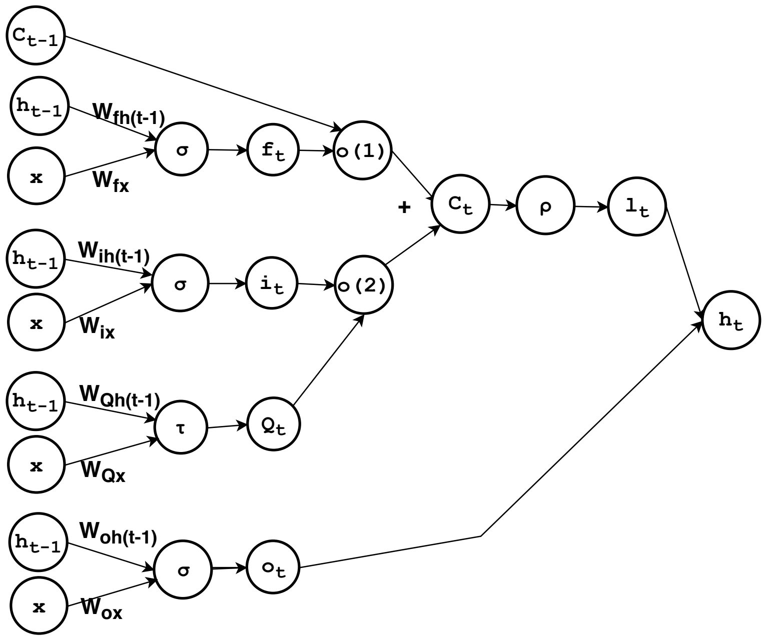

FIGURE 2.12: **Computational Graph of the LSTM Neuron**

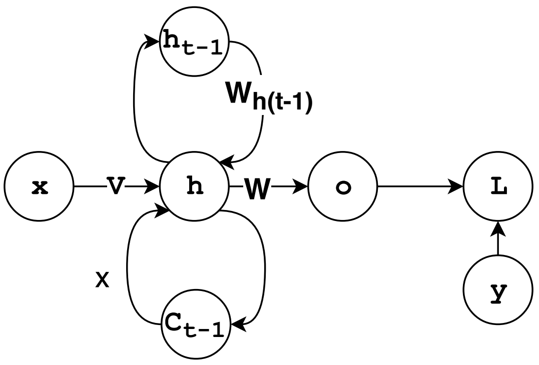

FIGURE 2.13: **Simplified Computational Graph of the LSTM network**

2.3.2 Gated Recurrent Units

The Gated Recurrent Units (GRU) was introduced by Cho, et al. to address machine translation using an encoder and decoder RNN [9]. The GRU replaces the *forget* and *input gate* in the LSTM with a single *update gate*. The reasoning for doing so, is to create a neuron that is simpler to implement and compute [9]. The architecture of the GRU is presented in figure 2.14. In figure 2.14 $\mathbf{r_t}$ represents the output of the *reset gate* and $\mathbf{z_t}$ is the output of *update gate*. The other elements are consistent to those previously explained for the LSTM in section 2.3.1.

The output of the *update gate* is calculated as in equation (2.21). This gate determines by how much the previous neuron output $\mathbf{h_{t-1}}$ will be used to generate the current

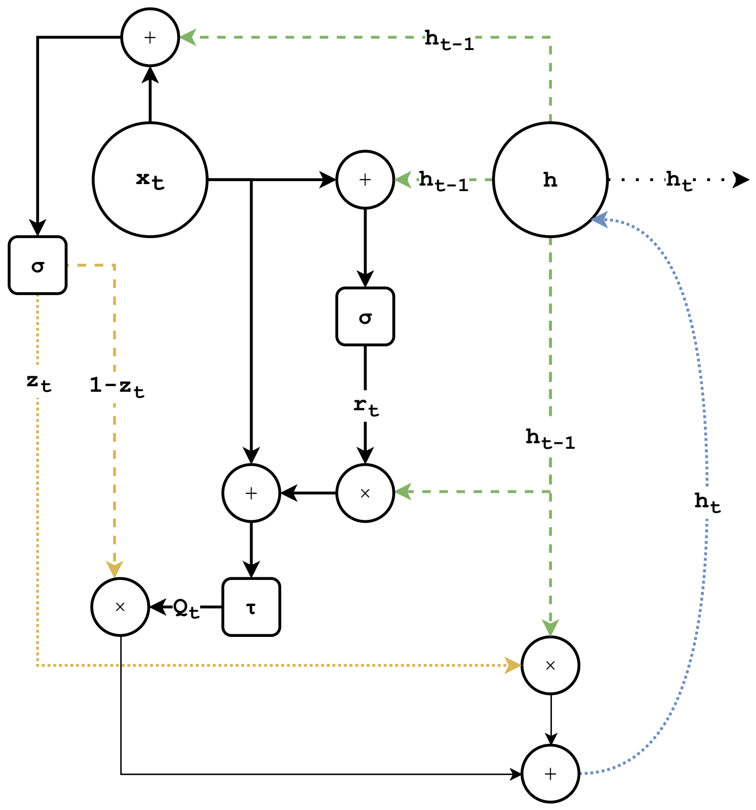

FIGURE 2.14: **Architecture of the GRU Neuron**

output $\mathbf{h_t}$. The update gate acts as the memory cell of the LSTM and can be regarded as an adaptive leaky-integration unit, which allows the network to retain long-term dependencies while maintaining stability [9].

$$\mathbf{z_t} = \sigma\big(\mathbf{W_{zx}x_t} + \mathbf{W_{zh-1}h_{t-1}}\big) \tag{2.21}$$

A *reset gate* is used to manage the propagation of the previous hidden state h_{t-1} of the neuron. When the reset gate outputs 0, only the current input is used to determine the current output of the neuron and the previous hidden state is ignored. Thus allowing the network to ignore any irrelevant information from the past [9]. The reset gates's output is calculated using equation (2.22) and the weighting of the reset factor is applied in equation (2.23) to produce the intermediate hidden state $\mathbf{Q_t}$.

$$\mathbf{r_t} = \sigma\big(\mathbf{W_{rx}x_t} + \mathbf{W_{rh_{t-1}}h_{t-1}}\big) \tag{2.22}$$

$$\mathbf{Q_t} = \tau\big(\mathbf{W_{Qx}x_t} + \mathbf{W_{Qh_{t-1}}}(\mathbf{r_t} \circ \mathbf{h_{t-1}})\big) \tag{2.23}$$

Finally the output of the neuron $\mathbf{h_t}$ is calculated as in equation (2.24). This demonstrates how the update gate and the intermediate state are weighted combined to produce $\mathbf{h_t}$ [9].

$$\mathbf{h_t} = (1 - \mathbf{z_t}) \circ \mathbf{h_{t-1}} + \mathbf{z_t} \circ \mathbf{Q_t} \tag{2.24}$$

The full computational graph of a single GRU node is presented in figure 2.15.

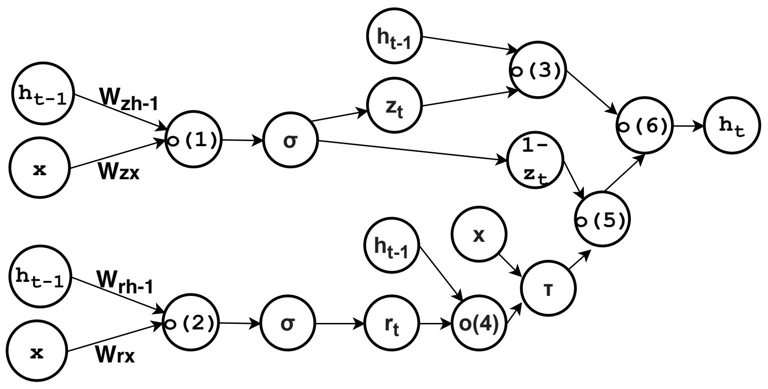

FIGURE 2.15: **Computational Graph of the GRU Neuron**

Figure 2.15 is a graphical representation of the GRU node's equations as the computational graph indicates the process flow of calculating the hidden state. A summarised version of this graph is presented in figure 2.16.

2.4 Bidirectional RNNs

First introduced in 1997 by Schuster and Paliwal, the Bidirectional RNNs (BRNNs) network architecture is configured to utilise information from the past, the present and the future patterns presented to it [43]. This is done by using two parallel hidden layers which form the bidirectional layer. The first hidden layer, referred to as the forward

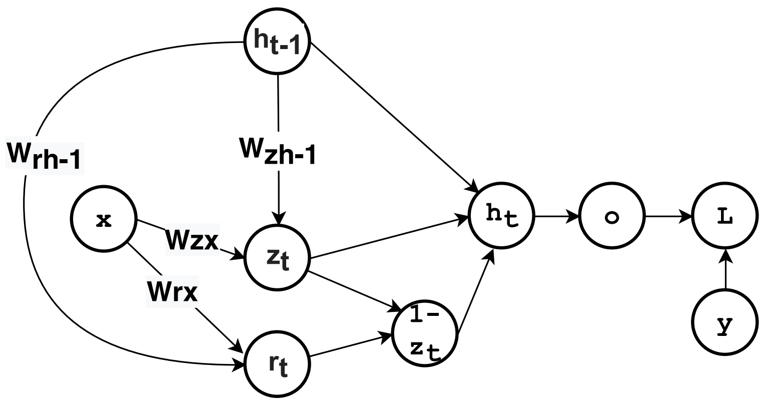

FIGURE 2.16: **Simplified Computational Graph of the GRU unit**

layer, receives the hidden states from the previous time steps as in the case of normal RNNs, The second layer, referred to as the reverse layer, however receives hidden states from future time steps. These future time steps can only be learned from once the whole input sequence has been presented to the network. Both these layers are then connected to the next layer, however they do not interact with one another [43].

The Bidirectional RNN configuration can be used with any of the RNN units presented in this chapter. As noted by the authors of the BRNN, updating the network state would be the same as other networks as only the order in which the input sequence is presented to nodes is different. Over the course of training, the network learns the optimal look-back periods in both the forward and backward sequences to find the best relationship between them.

BRNNs were developed with language models in mind where the current text is dependant on the context of future text yet to be presented. More specifically, the authors applied this architecture successfully to the TIMIT[1] phoneme database. In the work of Schuster and Paliwal, it was indicated that the Elman RNN achieved a recall rate of 64.32% whereas the BRNN achieved 68.53%, thus substantially outperforming the Elman RNN.

The computational graph of the BRNN is presented in figure 2.17. Here the forward layer and its components are presented using the subscript f and the reverse layer by subscript r.

[1]TIMIT is a corpus of English text aligned with audio recordings each being read. See https://catalog.ldc.upenn.edu/LDC93S1

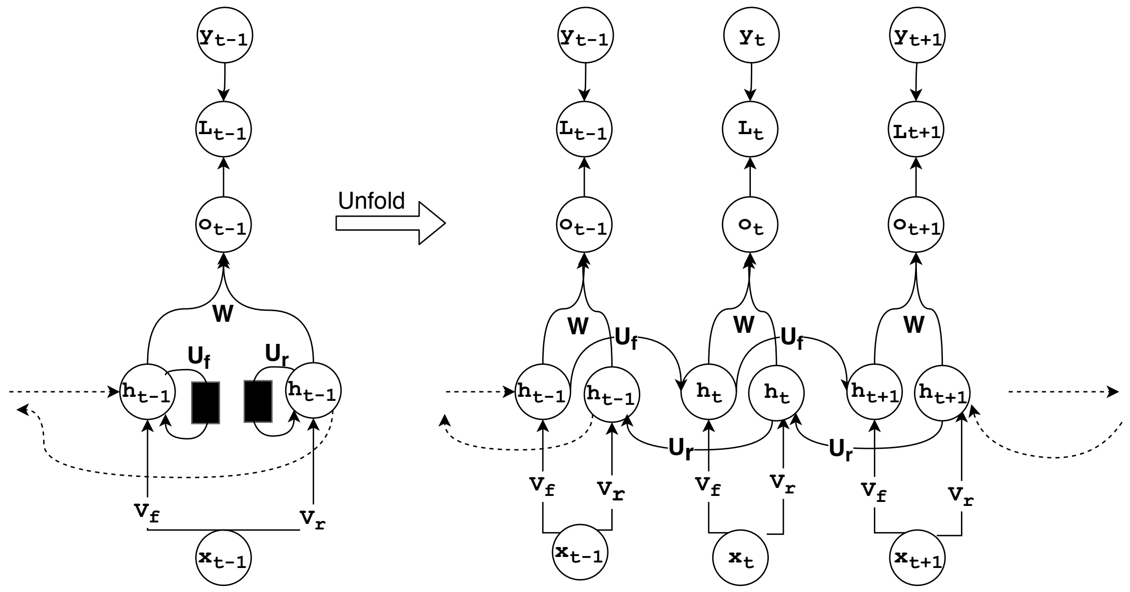

FIGURE 2.17: **Computational Graph of the Bidirectional Neural Network Architecture**

BRNNs are trained similarly to other RNNs. However if a variant of back-propagation through time (BPTT)[2] is used, the forward and backward weights cannot be updated simultaneously as the forward and backward layers do not directly interact. Initially, the BRNN is unfolded into both the forward and backward states. Once unfolded both the forward pass is run as well as the backwards pass. After both the forwards and backwards passes have been completed, the weights of each layer is updated. When the forward and backward states are not known at the current time step, the unknown derivatives are set to zero [43].

2.5 Summary

In this chapter the basic foundation of artificial neural networks are discussed and expanded on by introducing recurrent neural networks (RNNs). A few variants of the RNN were presented namely the Elman RNN and Jordan RNN. RNN neurons, the LSTM and GRU, have also been discussed. The LSTM unit is presented in section (2.3.1). Lastly the Bidirectional RNN was presented in section 2.4 alongside the reasoning for utilising its architecture.

[2]An explanation of back-propagation through time is presented in section 4.5

RNN modules under investigation in this dissertation are limited to those discussed in sections (2.2.1), (2.2.2), (2.3.2) and (2.4). Future work may include investigating Peephole Augmented LSTMs [20], Connectionist Temporal Classification (CTC) in 2006 [23] and more as there are numerous variants of RNNs, each with their own advantages.

Chapter 3

Memory Retention of RNNs

Chapter 3 discusses a review of previous work. The previous work on Memory retention of RNNs reviewed in this chapter, will form the basis of the experiments conducted in later chapters.

3.1 Capacity of Neural Networks

According to Cover's Function Counting Theorem released in 1965 [13] as noted by many authors [8][19][14] a single Perception is able to recognise at most $2N$ linearly separable patterns, where N is the number of inputs to the Perception. For non-linearly separable input spaces the capacity is reduced to one pattern per neuron in the sense that a neuron would be able to distinguish whether input is of the stored pattern or not [35]. In non-linearly separable input spaces, a neuron represents a single decision boundary. When adding neurons to the hidden layer, the network combines decision boundaries to form more complex boundaries. Thus hidden layer neurons are required only when the data is not non-linearly separated.

Huang and Guang-Bin [29] formally proved that a two layer FFNN is able to correctly classify N patterns using $L_1 = \sqrt{(o+2)N} + 2\sqrt{\frac{N}{(o+2)}}$ neurons in the first layer and $L_2 = o\sqrt{\frac{N}{(o+2)}}$ neurons in the second layer where o is the number of neurons in the output layer. Using the formulas presented, the RNNs should be able to recognise at least the same number of patterns as a FFNN using the same architecture.

3.2 Retention metrics

To quantify the retention of patterns between different neural network architectures, the per parameter capacity metric is employed. The per parameter capacity metric is used by Collins et al. [12] to compare the retention of patterns in RNN. The per parameter capacity ratio comprises of dividing the number of correctly classified patterns by the number of parameters in the RNN as is presented in equation (3.1).

$$\text{Capacity Ratio} = \frac{\#\text{Patterns Retained}}{\#\text{Parameters In Network}} \tag{3.1}$$

The higher the per parameter capacity ratio, the more effective the RNN is at utilising the parameters allocated in the RNN's architecture.

3.3 Memory Retention of Jordan and Elman RNN's

The number of studies relating to the sequence length retention in RNNs are limited. The problem with RNN's capacity is not only how many patterns are retained, but also for what time span over the observed set these patterns can be recalled. Furthermore it has been shown by Siegelmann and Sontag in 1992 that any commutable function can be represented by a finite RNNs [44].

3.4 Memory Retention of LSTMs

In the original LSTM paper, Hochreiter and Schmidhuber ran six experiments to test the LSTM's capacity [26]. Each experiment relied on a synthetically constructed benchmark, which was designed to demonstrate specific aspects of model retention. Hochreiter and

Schmidhuber used the embedded Reber grammar in the first experiment, which tests short time lags to demonstrate the benefit of the LSTM's output gates [26]. Thereafter, the author's next experiment consisted of noise free constructed sequences consisting of up to 1000 time lags [26]. The third experiment conducted consisted of applying noise to the *2-Sequence* problem [1]. The *2-Sequence* problem consists of classifying a presented sequence pattern as being one of either two sequences, during which the presented sequence may only be a partial representation of the full sequence. In the third experiment a 10000 sequences were used with a maximal lag of 100 [26]. The fourth experiment consisted of teaching the network to add a sequence of numbers until some marker was encountered in the sequence [26]. Lags of length up to 500 were used in the fourth experiment with 2000 sequences [26]. Experiment five consisted of the same structure as experiment four, except it was a multiplication problem using a lag of 50 and 2000 sequences [26]. The last experiment set out to test temporal ordering by generating input signals which appear at different lags, but which retained information when observed in a specific order in the sequence. Input lengths between 100 and 110 were used with 4 classes that could be output [26]. A maximum of 2560 sequence lengths were used [26]. Hochreiter and Schmidhuber concluded in each experiment that the LSTM nodes outperformed each other RNN considered in the experiments conducted.

Another paper Hochreiter contributed to, alongside Schmidhuber and Jürgen, looked at solving non-trivial time lag problems [27]. The authors specified the difference between trivial and non-trivial tasks. Tasks that fall into the trivial class include the "2-Sequence", latch, parity problems as well as Tomita grammars [27]. These problems could be solved using a simple random search through the weight space of the applied neural networks [27]. Non-trivial problems were those that required many free parameters and/or high weight precision to be solved [27]. Hochreiter et. al used the adding problem as a non-trivial task to test the LSTM. The adding problem requires the network to add all values seen until some stop character was reached [27]. It was shown that the LSTM could correctly produce the correct results in 2557 out of 2560 cases for sequences ranging up to 1000 time lags [27].

[1]The *2-Sequence* problem was first presented by Bengio et al [5]

The approach used by Collins et al. [12] is to generate unique stochastic binary input-output pairs. From the experiments executed, Colins et al. found that for numerous RNNs investigated, namely; the Elman, LSTM, GRU. Update-Gate RNN and more, RNN neurons are able to retain 5 bits per network parameter.

3.5 Memory Retention of GRUs

With regards to memory capacity of the GRU, the only study that provided some insight was by Collins et al. in 2017 [12]. In this study the authors noted that the memory retention ratios of the Elman RNN's, LSTMs and GRUs are the almost the same with the Elman RNN being able to retain more information if the same number of parameters were used across the network. However, this paper noted that for deep architectures and more complex function spaces, the GRU is much more trainable than the LSTM or Elman RNN.

Chapter 4

Training Algorithms

There are many gradient based algorithms for training neural networks. Each algorithm provides a different method for updating the weights that parameterise the neural network based model. The best training algorithm to use in practice depends on both the underlying data set and the architecture under consideration. In this chapter the set of training algorithms used in the experiments are discussed. The training algorithms presented, serve to ensure that the results produced were not skewed by a single algorithm that failed to train an otherwise superior RNN architecture.

Within the equations contained inside of this chapter θ represents the weights within the NN being optimised, η the learning rate, J is the loss function being optimised, x represents pattern presented to the network, y the target vector, v the error gradient vector, i the current iteration.

4.1 Gradient Descent

4.1.1 Algorithms

4.1.1.1 Batch Gradient Descent

Batch gradient descent, which is also the first and most simplest form of gradient descent, computes the gradient between the error function and each entry in the data set during an epoch [41]. This implies that the whole data set needs to be considered for one error propagation through the weights of the NN being trained. The propagation of batch gradient decent is presented in equation (4.1), the notation of which was taken from Ruder et al. [41]. Adding to the previously stated parameters, ∇_θ represents the sum of errors between the expected target and the output produced.

$$\theta = \theta - \eta \nabla_\theta J(\theta) \tag{4.1}$$

4.1.1.2 Stochastic Gradient Descent

Stochastic gradient descent (SGD) follows a more robust methodology by updating the weights of the NN for each element presented to the neural network [41]. By updating the weights when an element is presented to the network, SGD allows the NN to learn as patterns are presented to it, thus the data set can be much larger than the memory available of the machine on which the NN is trained. SGD also has the ability to jump out of local minima which would have trapped a NN trained using Batch gradient descent. The reason SGD can get out of local minima better than Batch gradient descent is that the variance of the gradient update is larger. Since the variance is larger, SGD is able to explore more of the feature space. However, the larger variance in the gradient update may also cause that SGD to overstep solutions. To handle the overstepping of optima, gradually reducing the learning rate over time, has been shown to increase the rate of convergence on a solution. The equation for SGD, is presented in equation (4.2) and follows the notation used by Ruder et al. [41]. In equation (4.2) i represents the current pattern index being presented to the network, which indicates the update happens at each iteration.

$$\theta = \theta - \eta \nabla_\theta J(\theta; x^{(i)}; y^{(i)}) \tag{4.2}$$

4.1.1.3 Mini-batch Gradient Descent

Mini-batch gradient descent combined both Batch and Stochastic gradient descent by using a sub-sample of size n of the training set to update the weights of the NN in question [41]. The training process is repeated until all sub-samples of the training set is presented to the NN which forms an epoch. In using Mini-batch gradient descent, the variance in the gradient update is reduced leading to better rates of convergence, as well as enabling the ability to matrix algebra to optimise the computation of the error gradient. Borrowing from Ruder et al. [41], the computation for Mini-batch gradient descent is presented in equation (4.3).

$$\theta = \theta - \eta \nabla_\theta J(\theta; x^{(i;i+n)}; y^{(i;i+n)}) \tag{4.3}$$

4.1.2 Momentum

As mentioned in section 4.1.1, when computing gradients step-wise as in the case of SGD or Mini-batch gradient descent, the variance in the gradient update can cause the NN to step over possible optima. Gradient based training algorithms also struggle with increasing the rate of convergence when the optima is clear, and reducing the rate of convergence when the feature space is more stochastic.

A common used strategy to improve the performance of gradient based training algorithms is to introduce a momentum term [38]. A momentum term increases the rate of convergence while decreasing the sporadic update in the error gradient propagated back in the NN. Momentum uses the previous error gradient vector v at $t-1$ weighted down by a scalar $\gamma \in (0,1)$ in the current update of the weights in the network. The update is then calculated as in equations (4.4) and (4.5).

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta; x^{(i;i+n)}; y^{(i;i+n)}) \tag{4.4}$$

$$\theta = \theta - v_t \tag{4.5}$$

4.2 Nesterov Accelerated Gradient

Yurii Nesterov proposed a method for utilizing the momentum term γ during the back propagation phase of the gradient descent algorithm that greatly improves convergence of NNs. The Nesterov gradient descent algorithm computes an expected (approximate) gradient update by taking in consideration the previous gradient [37][41]. The updates for the gradient are presented in equations (4.6) and (4.7):

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1}; x^{(i;i+n)}; y^{(i;i+n)}) \tag{4.6}$$

$$\theta = \theta - v_t \tag{4.7}$$

Nesterov Accelerated Gradient (NAG) thus first propagates the momentum term combined with the partial gradient which is approximated. Thereafter the gradient change at the current epoch is applied. By doing so, the weights are first propagated into the direction of the previous epoch's momentum and is then corrected into the right direction using the current gradient update [37].

The idea behind the NAG algorithm is that it prevents the momentum from becoming unbounded and pulling the network out of the optima being exploited, by overshooting, causing the update to be more granular.

The NAG algorithm specifically increased the performance of RNN's on various task [4], however the NAG algorithm is flawed in that, the same learning rate is applied when updating the weights of NNs.

4.3 RPROP

Resilient Propagation (RPROP) was introduced in 1993 by Riedmiller and Braun to address problems with learning rate propagation [39]. Riedmiller and Braun reasoned that the magnitude of the gradient changes for each layer of the neural network. By adjusting the learning rate considering only a single gradient, overshooting of good local positions often occur [39]. All learning rates are set to some global learning rate and thereafter gets adjusted relatively. When the signs of the current back-propagating gradient and the previous are the same, increase the learning rate so as to increase the step size of the gradient update to induce exploration. When the signs differ, decrease the learning rate to increase exploitation [39]. Calculation of the gradient for each parameter j at time step t will thus be as presented in equation (4.8). The gradient propagation is presented in equation (4.9) and the learning rate adapts using (4.10) with the learning rate η constraint as in equation (4.11).

$$g_{t,j} = \nabla_{\theta_t} J(\theta_{t,j}) \tag{4.8}$$

$$\theta_{t+1,j} = \theta_{t,j} - \eta \cdot g_{t,j} \tag{4.9}$$

$$\eta = \begin{cases} \eta^+, & \text{if } g_{t,j} \cdot g_{t-1,j} > 0 \\ \eta^-, & \text{otherwise} \end{cases} \tag{4.10}$$

$$0 < \eta^- < 1 < \eta^+ \tag{4.11}$$

4.3.1 RMSPROP

Problems arose with RPROP when using batch learning. The magnitude of gradients for observations in the same batch would cancel one another, stagnating training. In 2012 Tieleman and Hinton et al. devised the RMSPROP algorithm [24]. RMSPROP (Root Mean Squared Error Propagation) keeps the moving average of the squared gradient for each weight which is then used to update the weights as in equations (4.12) and (4.13).

In equations (4.12) and (4.13), MSE represents the mean squared error function as is defined in section 5.1 and ϵ is an arbitrarily small term to avoid division by zero, which is set to a value smaller or equal to 0.00000001.

$$\text{MSE}[g^2]_t = 0.9\text{MSE}[g^2]_{t-1} + 0.1g_t^2 \tag{4.12}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\text{MSE}[g^2]_{t-1} + \epsilon}} g_t \tag{4.13}$$

4.4 Adam

Diederik Kingma and Jimmy Ba presented the Adam training algorithm in 2014 [34]. Adam is short for Adaptive Moment Estimation as it adapts learning rates for each weight as needed. It does this much in the same way as RPROP. For each weight it adapts the learning rates using the exponential average of the gradient and the squared gradient. The step size of these gradients are controlled by the parameters β_1 and β_2 respectively as it decays as needed. A caveat with this approach is that β_1 and β_2 are initialised close to 1 which introduces a bias towards 0. To alleviate this problem the bias estimates are first calculated and only thereafter bias-corrected estimates are determined. The calculation of the Adam optimiser is as presented in equation (4.14) as defined by Kingma and Ba [34].

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{4.14}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{4.15}$$

$$\hat{m}_t = \frac{m_t}{1 - (\beta_1)^t} \tag{4.16}$$

$$\hat{k}_t = \frac{k_t}{1 - (\beta_2)^t} \tag{4.17}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{s}_t} + \epsilon} \hat{m}_t \tag{4.18}$$

In equation (4.14) m_t represents the first moment state and k_t the second. $\hat{m}_t$ is the first bias correction and $\hat{k}_t$ the second. The last step in the equation is the gradient propagation through the network. Kingma and Ba [34] recommend values of 0.9 for β_1 and 0.999 for β_2.

4.5 Back Propagation Through Time

Back Propagation Through Time (BPTT) is the extension of traditional gradient decent based algorithms to handle the recurrently defined models that make up RNNs. BPTT is used to train RNN's by propagating errors and updating weights over time [48].

BPTT functions by unrolling all time steps it received, meaning that it keeps a "copy" of the neural network, input and output for each time step that it is unrolled for [48]. By doing so, errors are accumulated for each time step and then propagated back through each time step. The unrolling of the network is usually limited to a set look-back interval. Unrolling of the network incurs much more computational cost. The longer the look-back period, the more the network is unrolled, the longer training will take.

Here expected and predicted output is defined as o_t and y_t at time t respectively. The matrices of all expected and predicted output are O and Y with the maximum number of time steps presented as T. From this representation, an arbitrary loss function is represented by ℓ as in equation (4.19):

$$\ell(Y,O) = \sum_{t=1}^{T} \ell_t\big(y_t, o_t\big) \tag{4.19}$$

The steps involved in the BPTT algorithm is presented in equations (4.20), (4.21), (4.22). In the BPTT equations, activation functions are represented by f. Furthermore, the weight matrices are represented by $\mathbf{W}$, $\mathbf{U}$ and $\mathbf{V}$ between the recurrent state s, input x and the output vectors o.

$$\begin{aligned}
\frac{\partial \ell}{\partial \mathbf{W}} &= \frac{\partial}{\partial \mathbf{W}} \sum_{t=1}^{T} \ell_t\big(y_t, o_t\big) && (4.20)\\
&= \frac{\partial}{\partial \mathbf{W}} \sum_{t=1}^{T} \ell_t\big(y_t, g(\mathbf{V} f(\mathbf{U} x_t + \mathbf{W} s_{t-1}))\big) && (4.21)\\
&= \sum_{t=1}^{T} \frac{\partial \ell_t}{\partial g} \frac{\partial g}{\partial f} \frac{\partial}{\partial \mathbf{W}} f(\mathbf{U} x_t + \mathbf{W} s_{t-1}) && (4.22)
\end{aligned}$$

When considering that $s_{t_1} = f(\mathbf{U}x_{t-1} + \mathbf{W}s_{t-2})$, the use of the chain rule and the product rule are used for each step of the unrolling of the RNN.

Chapter 5

Experiment Performance Metrics

In this chapter an example is presented to illustrate that the traditional mean squared error loss function, while effective for training the underlying model, is not sufficient for accessing the memory retention ability of the model. For this reason, the F1-score is applied to determine true retention.

This chapter follows the format in which the mean squared error (MSE) function is presented in section 5.1. Section 5.2 describes how retention is measured within this dissertation. Lastly, section 5.3 describes how measurements are made when the experiment is structured as a regression problem statement.

5.1 Mean Squared Error

The mean squared error is defined as in equation (5.1). In equation (5.1) o_t is the expected output and y_t is the predicted result. Furthermore N represents the number of elements in the output vector as the length of y_t and o_t would be equivalent.

$$\ell_{t_{mse}} = \frac{1}{N} \sum (o_t - y_t)^2 \tag{5.1}$$

The MSE function can be utilised as the objective function the NN is trained to minimize. The MSE can be utilised when working with both regression and classification problems, though the former is more common. One of the largest critiques of the MSE loss function points to the use of squaring each term. By using the square function, the MSE weighs larger discrepancies between o_t and y_t values higher. In doing so this leads the variance of the error to increase drastically if there are many outliers in the training set [7].

For the experimental formulation, the shortfalls of MSE is addressed by not introducing outliers into the training data sets.

5.2 Measuring Memory Retention Performance

To enable measurement of the patterns recalled, the confusion matrix is utilised to calculate precision and recall values of the trained models [46]. A confusion matrix has four elements, which shows the number of patterns correctly classified and those incorrectly classified for each possible output class. Recall specifies how many of the patterns observed were correctly classified as a specific class by also factoring in if all instances of the specific class was correctly classified and is defined as in equation (5.2) [15].

True positives are the number of patterns which are correctly classified as the expected class which is also referred to the positive cases. True negatives are the number of patterns which correctly state that the prediction is not the positive case. False negatives are the number of patterns for which the expected class is the positive case, however the model incorrectly produced the opposite. False positives are the cases in which the model produces the positive case of a class when the negative case is expected.

$$recall = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}} \tag{5.2}$$

Precision is the metric that indicates how well our model is at discriminating data points [15].

$$precision = \frac{\text{true positives}}{\text{true positives} + \text{false positives}} \tag{5.3}$$

Precision and recall is forever in a trade-off when the algorithm does not overfit the input space [15]. An example of the trade-off in which recall is increased is when increasing the number of correctly classified instances of a specific class may have the effect that more false positives increase. In the case that precision is preferred, an example would be when the number of false positives decrease, however the number of false negatives increase which reduces recall. To balance this trade off between precision and recall, the F1-score is used.

$$F1 = \frac{2 * (\text{precision} * \text{recall})}{\text{precision} + \text{recall}} \tag{5.4}$$

In the experimentation conducted in this dissertation, the training of the neural network is only halted once it has achieved a F1-score of 1, which indicates that the relationship between input and output has been retained.

5.3 Structuring Regression as a Classification task

Within this dissertation an investigation to compare the effect of structuring the experiments as either a regression or classification task will be investigated. However, the main metric to measure retention is the F1-score as presented in section 5.2, which only supports classification tasks. For this reason the regression experiments need to be structured as classification tasks as well. To achieve this, one node is used as the output node using some activation function. The output is scaled to be between $[0, 1]$. The output of the network is then compared to the expected categories using euclidean distance. The category with the closest euclidean distance is then chosen as the category which the neural network predicted.

As an example, consider the case in which three classes are to be predicted using the sigmoid activation function. The first class would encompass the range $[0, 33]$, the second class range $(0.33, 0.66]$ and the third $(0.66, 1.0]$. Thus binning is applied to structure a regression task as a classification task.

Chapter 6

Investigation of the Retention Length and Memory Capacity of Recurrent Neural Networks

To test the memory retention capacity and length of RNN's, multiple experiments will be conducted. The structure of these experiments will be discussed in this chapter.

Within this chapter the inability of the RNNs to learn is quantified by measuring the decrease in loss of the network. When the network's loss-rate plateaus for a period of 100 epochs, it is deemed that training has been exhausted. After the training of the RNN is stopped, the retention is determined by the $F1$-score.

6.1 Experiment Design

The experiments discussed in this section are inspired from the original LSTM paper by Hochreiter and Schmidhuber [26] and also the work done by Collins et al. [12] which is discussed in section 3.4.

6.1.1 Constraints Ensuring Fairness of Comparison

In order to compare RNN's capacity, the total number of parameters in all networks for a single experiment are made equal. Using a equal number of parameters is the standard practice to ensure fairness in comparison [3][12][31][49]. With the LSTM and GRU being much more complex than the Jordan and Elman RNNs, more nodes will be required for the latter models to ensure an equal number of parameters are present. In addition it must be ensured that all networks being compared have the same number of hidden layers. If one RNN is given more layers than another, the former has the benefit of additional abstraction which has the potential to unnecessarily skew results.

A RNN node has multiple internal parameters (weights) which facilitates recursive behaviour. These internal parameters are not to be confused between the edges (weights) that connect nodes from one layer to the next. A single LSTM node consists of 12 parameters, a GRU node consists of 9 parameters and the Elman RNN consist only of 3 parameters per node. To be able to compare the performance of the Elman RNN to the LSTM network, for each LSTM node, 4 nodes in the corresponding layer needs to be included for the Elman to ensure a fair comparison. To compare networks consisting of LSTM's to those of GRU's, the same concept applies, however the number of parameters in corresponding layers need to be divisible by both 12 and 9 such as 36. The calculation of the per parameter capacity ratio is presented in section 3.3.

Unfortunately, the Jordan RNN's increase the number of parameters depending on the number of nodes in the next layer. This makes comparison between this network and the other more difficult. However, if the performance of the network is normalised to the performance per parameter, RNNs can be compared as long as the RNNs have the same number of hidden layers.

6.1.2 Spearman Rank Correlation Coefficient

The Spearman Rank Correlation Coefficient (SRCC) is used to measure the strength of a linear relationship between two variables. In this dissertation, the effects of various parameters are correlated to the outcome of experiments. The formula for the SRCC is presented in equation (6.1). In equation (6.1) d_i represents the difference between the ranks of the corresponding pattern in each set being compared and n represents the number of observations. Rank is assigned from 1 to n where 1 is the highest rank assigned to the highest value in the set being ranked and n to the lowest value in the current set being ranked. After both sets are ranked independently, the ranked value r_{spearman} is calculated as in equation (6.1). The resulting value of r_{spearman} is in the range of $[-1, 1]$. A value of 1 indicates a perfect positive relationship, whereas, -1 indicates a perfect negative relationship.

$$r_{\text{spearman}} = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)} \tag{6.1}$$

6.1.3 Activation Functions to be tested

As stated by Collins et al.[12] the activation functions used have a direct effect on the capacity of the parameters. The Rectified Linear Unit (relu) activation function reduced memory capacity, however using it increases trainability of the networks. The same observation is made within this dissertation.

In all cases it is better to be able to train a network at the cost of having more parameters and a reduction in memory retention per parameter. In other words, if a small network is unable to infer any information from the training set, but a larger network is, it is better to use the larger network as long as overfitting is accounted for, and the computational cost of running the model is still feasible for the target application area.

The activation functions considered are contained within table 6.1. The choice of activation functions are due to a combination of popularity of use and availability within the Keras package [10]. In table 6.1 x refers to the input scalar or vector passed to the activation function.

Activation function	Equation
elu (Exponential Linear Unit)	$= \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x < 0 \end{cases}$
linear	$= x$
relu (Rectified Linear Unit)	$= \max(0, x)$
hard sigmoid	$= \begin{cases} 0 & \text{if} x < -2.5 \\ 1 & \text{if} x > 2.5 \\ 0.2x + 0.5 & \text{if} -2.5 <= x <= 2.5 \end{cases}$
softmax	$= \frac{e^x - max(x)}{\sum(e^x - max(x))}$
softplus	$= log(e^x + 1)$
softsign	$= \frac{x}{(abs(x)+1)}$
sigmoid	$= \frac{1}{1+e^{-x}}$
tanh	$= \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$

TABLE 6.1: Activation functions considered in experiments

6.2 Experiments

The experiments conducted are outlined within the subsections contained within. The experiment formulation for sequence length retention is presented in section 6.2.1 which further consists of a regression and classification version in sections 6.2.1.1 and section 6.2.1.2 respectively. To determine the number of patterns which can be retained by RNNs the second experiment is proposed in section 6.2.2. The pattern retention experiments are also formulated in a regression and classification component which are presented in section 6.2.2.1 and 6.2.2.2. Lastly, the frame of retention experiment formulation and justification of using regression only is presented in section 6.2.3.

6.2.1 Experiment 1: Length of Retention

6.2.1.1 Version 1: Regression

The goal of this experiment is to determine the number of time steps (which will be referred to as length) the networks are able to retain patterns over a number of epochs. To investigate length of retention, a counting problem is employed. A sequence of binary numbers is generated to form patterns of length b_n. During time step t a single bit is presented to the network. The network is tasked to count the number of 1's between observed 0's and to output the result. The length between 1's will gradually be increased until the network is unable to learn. The length desired at the output state must be within the range $[0, 1]$ to allow for the use of different activation functions. Thus if the count is 3, the expected output will be $\frac{1}{3}$ or 0.333333.

Example: Given $b_n = 3$ the mapping between input to output over number of time steps is presented in table 6.2. In table 6.2 $T1$ represents the first time step, the input and the expected output is shown in the corresponding row.

Time step	Input	Output
T1	1	1.0000
T2	0	0.0000
T3	0	0.0000
T4	0	0.0000
T5	1	0.5000
T6	0	0.0000
T7	1	0.3333
T8	0	0.0000
T9	0	0.0000

Table 6.2: Experiment 1: version 1 example mapping

6.2.1.2 Version 2: One-hot encoding classification

To study the effects of the way the modelling problem is presented, experiment 6.2.1.1 is repeated with the modification that the output is represented as a one-hot encoded sequence. Thus each time the maximum count increases, another node is added to the output layer. The output node with the largest activation is then chosen to represent the count predicted by the network.

Example: Given $b_n = 3$ the mapping between input to output over number of time steps is presented in table 6.3. In table 6.3 $T1$ represents the first time step, the input and the expected output is shown in the corresponding row.

Time step	Input	Output
T1	1	[1.0000, 0.0000, 0.0000]
T2	0	[1.0000, 0.0000, 0.0000]
T3	0	[1.0000, 0.0000, 0.0000]
T4	0	[1.0000, 0.0000, 0.0000]
T5	1	[0.0000, 1.0000, 0.0000]
T6	0	[0.0000, 1.0000, 0.0000]
T7	1	[0.0000, 0.0000, 1.0000]
T8	0	[0.0000, 0.0000, 1.0000]
T9	0	[0.0000, 0.0000, 1.0000]

TABLE 6.3: Experiment 1: version 2 example mapping

6.2.2 Experiment 2: Number of Patterns Retained

6.2.2.1 Version 1: Regression

Capacity is defined as the number of patterns that can be recognized. During a single experiment, a vector of random real numbers are generated as possible input, which map to another vector of randomly generated real numbers, as the target vector. To ensure that no underlying pattern exists between the sets, we apply a Spearman Rank

Correlation [16]. The Spearman Rank Correlation generates a correlation coefficient ρ and a probability value p. For each set generated, the set is accepted if $p > 0.05$ or $\rho \in [-0.1, 0.1]$ which indicates that no pattern exists in the set.

Example: Given the number of elements is 3, the mapping between input to output is presented in table 6.4. In table 6.4 $T1$ represents the first time step, the input and the expected output is shown in the corresponding row. Table 6.5 demonstrates that there is not a time dependence.

Time step	Input	Output
T1	0.5000	0.3000
T2	0.1000	0.7000
T3	0.3000	0.2000

Table 6.4: Experiment 2: version 1 example mapping

6.2.2.2 Version 2: One-hot encoding classification

In the same vein as experiment 1, a classifications variant of the experiment is also used. Here, for each pattern that should be retained, a corresponding output node exists. When the pattern is observed, the network should assign a higher probability to that node, which is also used in the second part of the first experiment in subsection 6.2.1.2.

Example: Given the number of elements is 3 a possible set could be as is presented in table 6.5.

Time step	Input	Output
T1	0.5000	[1.0000, 0.0000, 0.0000]
T2	0.1000	[0.0000, 1.0000, 0.0000]
T3	0.3000	[0.0000, 0.0000, 1.0000]

Table 6.5: Experiment 2: version 2 example mapping

6.2.3 Experiment 3: Frame of Retention

Frame of retention is defined as the number of patterns of increasing sequence length the neural network is able to retain. Building off of experiments 6.2.1 and 6.2.2, this experiment is also formulated as a counting problem. Given a sequence of numbers, the network is tasked to count the number of times it has encountered the element presented to it at each time step. Stochastic sequences will be presented to the network. After the network's loss function has plateaued, it's frame of retention is determined. If the network is able to retain all patters in an experiment, another element is included into the set of possible integers and a new sequence is generated. This is repeated until the network is unable to retain the count of each element. The maximum count will be specified ahead of the experiment and will be used to control the maximum number of times an element may appear in the sequence presented to the network. Scaling of the elements will be done using the maximum number of elements to identify plus 1 and the output will be scaled using the maximum count plus 1. The formula discussed to scale input and output is defined in equation (6.2). In equation (6.2) x represents a single element within a set of real numbers, $\forall x$ indicates for all elements within the set and x_{scaled} is the scaled value produced for the current element considered in the set.

$$x_{\text{scaled}} = \frac{x}{\max(\forall x) + 1} \tag{6.2}$$

Example: Given the number of elements is 3 we will generate $\{[[[1]], [[1]], [[1]], [[3]], [[1]], [[2]], [[2]], [[3]], [[1]], [[1]]]\}$ and expect the network to output $[[[1]], [[2]], [[3]], [[1]], [[4]], [[1]], [[2]], [[2]], [[5]], [[6]]]$. After scaling the sequence will be $\{[[[0.25]], [[0.25]], [[0.25]], [[0.75]], [[0.25]], [[0.5]], [[0.5]], [[0.75]], [[0.25]], [[0.25]]]\}$ with the output being $[[[0.14]], [[0.29]], [[0.43]], [[0.14]], [[0.57]], [[0.14]], [[0.29]], [[0.29]], [[0.71]], [[0.86]]]$ given maximum number of elements is 3 and maximum count is 6. This example mapping is also demonstrated in table 6.6.

Time step	Input	Output
T1	0.25	0.14
T2	0.25	0.29
T3	0.25	0.75
T4	0.75	0.14
T5	0.25	0.57
T6	0.5	0.14
T7	0.5	0.29
T8	0.75	0.29
T9	0.25	0.71
T10	0.25	0.86

Table 6.6: Example of Experiment 3: Frame of retention

6.2.4 Format of each Experiment

A single experiment cycle will consist of training on the training set using a training algorithm as discussed in chapter 4, a specific activation function (section 6.1.3) and a fixed architecture. The performance on the test set will then be recorded. Training is only stopped when the error produced by the neural network converges and does not improve over 100 epochs. There is no upper limit on the amount of epochs for which training will be done.

For each of the discussed experiments, the training set, a 100 instances of each pattern that needs to be recognised will be generated and for the test set 10 samples for each pattern. This is done to balance computational cost while still allowing the network to train on a sufficient number of data points. During training the samples are fed in at a batch size of 10. Training is stopped when the loss of the neural network does not improve over 100 epochs. It is important to note that a sequence is seen as one sample. Thus if the sequence is set to 3, each of the 300 vectors will be shown to the RNN.

Learning rate starts at 0.9 and thereafter is set to decay when the network plateaus after 10 epochs at a rate of 0.01 with the smallest learning rate being 0.0000000000001.

6.2.5 Implementation Details

The experimental work within the dissertation was implemented using a python library called Keras utilising a Tensorflow back-end [1][10]. Keras is used as it specifies a simple interface for creating NN architectures.

All parameters for the implementation steps are kept at Keras default such as the learning rate and momentum factors since these parameters originate from the original papers in which each algorithm is implemented such as the Adam training algorithm [10]. For the LSTM nodes, the forget gate is reset after each batch as done by Akash Singh [45].

Chapter 7

Results

This chapter serves to present the results of the experiments described in chapter 6. In this chapter three types of experiments are investigated namely: sequence length retention, number of patterns that could be retained (referred to as the capacity of the network) and in combining the sequence length with number of patterns experiments, the frame retention rate per parameter experiment is formulated.

7.1 Experiment 1: Length of Retention

In this section the results of the length of retention experiment as described in section 6.2.1 are presented. In short, this experiment sets out to determine the sequences length that can be retained relative to the number of parameters in RNNs and how this relationship is effected by factors such as network size, activation function choices, and the training algorithm used. To investigate the effect of the problem statement formulation, each experiment is run as a regression and as a classification problem as stated in section 6.2.2.

The analysis starts off by investigating the overall performance of all RNNs, compared to one another on an aggregated level over each configuration, in section 7.1.1. Thereafter, the effects of using different activation functions evaluated is shown in section 7.1.2. Next in line is the investigation of the relationship between the per parameter sequence capacity and the number of hidden layers in section 7.1.3. Lastly the effect of using different training algorithms on the retention of RNNs is presented in section 7.1.4.

7.1.1 Average Per Parameter Sequence Length Retention for each RNN

The overall performance of each RNN is evaluated in this subsection to determine the overall behaviour of the various networks when applied to retain sequences of increasing length.

Figures 7.1, 7.2, 7.3, 7.4, 7.5, 7.6, 7.7 and 7.8 show the average sequence length retained with increasing number of parameters. For this specific experiment it is clear that after a certain point, increasing the number of parameters causes the sequence length retained by the network to plateau or even decrease regardless of classification or regression formulation. This result is counter intuitive as one would expect the network to be able to retain longer sequences the larger it is. The implication thereof is that simply increasing the network's size, does not necessarily increase it's ability to retain longer sequences and may even hinder sequence retention in some cases. Furthermore, this also implies that larger RNNs are more susceptible to local minima. The observed behaviour of larger architectures not necessarily increasing retention may be due to too many parameters interfering with the training of the RNNs increasing noise throughout the networks.

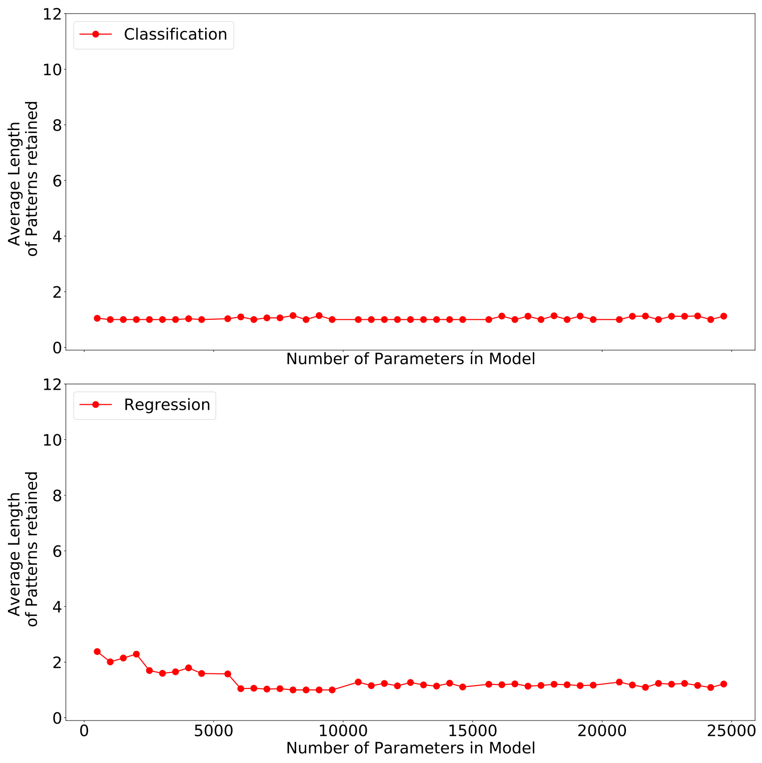

FIGURE 7.1: **Effect of the number of parameters on the ability to retain patterns for the bidirectional Elman RNN**

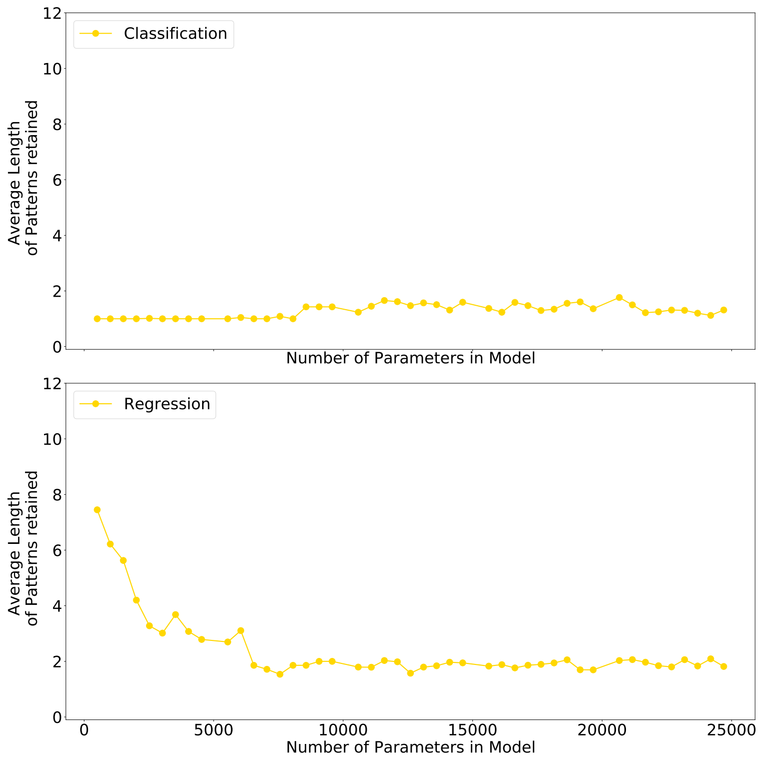

FIGURE 7.2: **Effect of the number of parameters on the ability to retain patterns for the bidirectional GRU RNN**

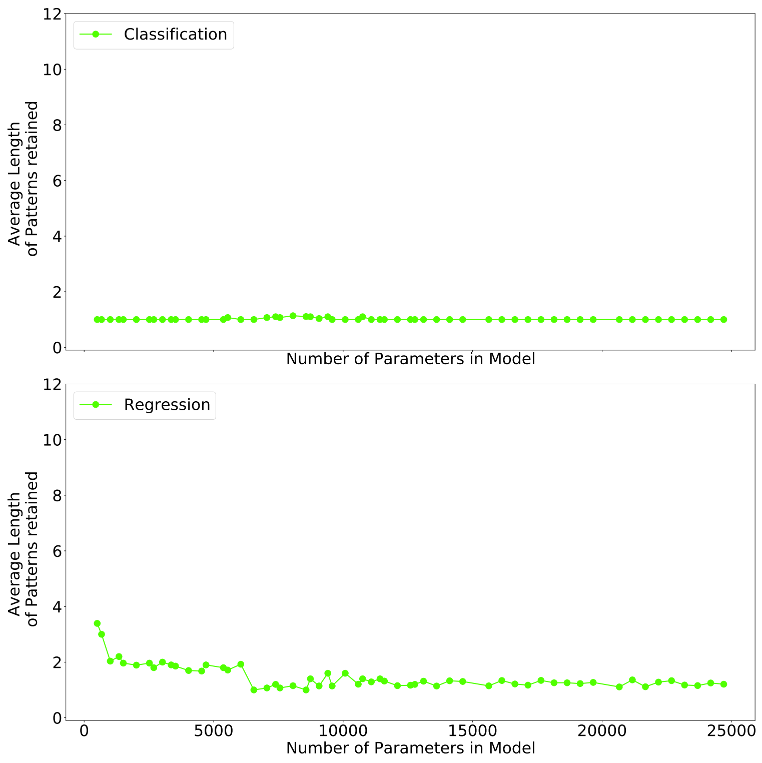

FIGURE 7.3: **Effect of the number of parameters on the ability to retain patterns for the bidirectional Jordan RNN**

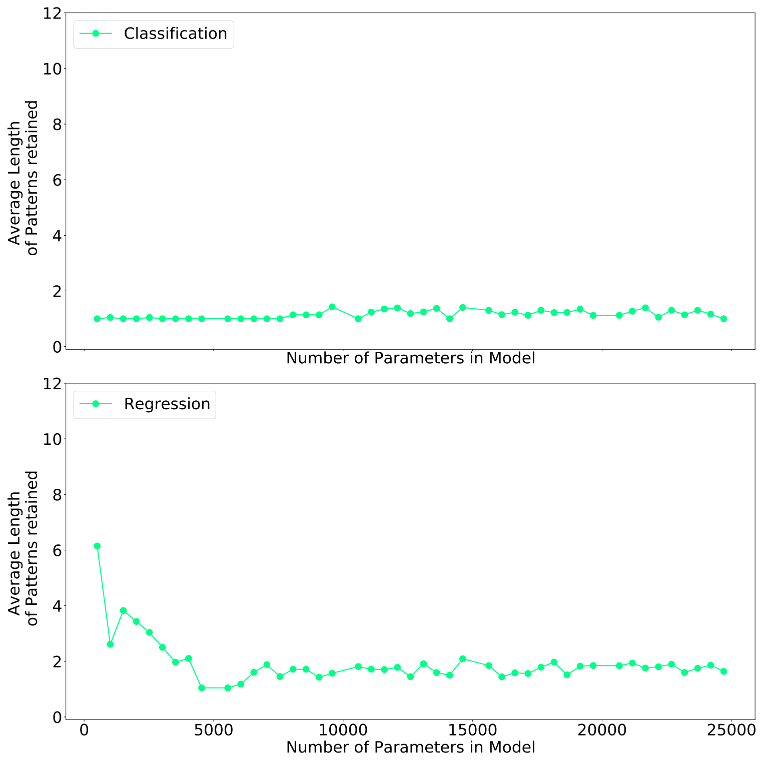

FIGURE 7.4: **Effect of the number of parameters on the ability to retain patterns for the bidirectional LSTM RNN**

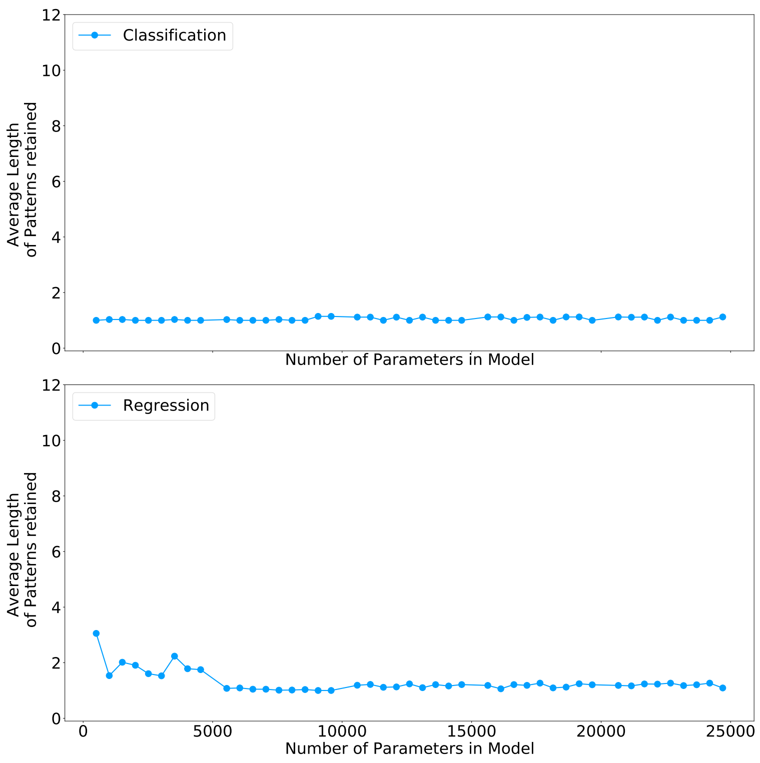

FIGURE 7.5: **Effect of the number of parameters on the ability to retain patterns for the Elman RNN**

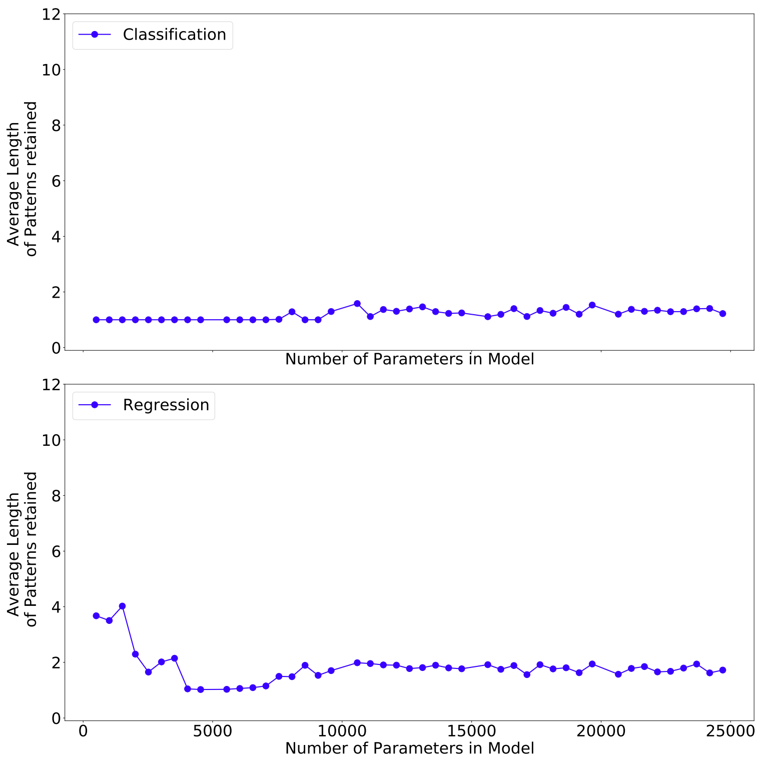

FIGURE 7.6: **Effect of the number of parameters on the ability to retain patterns for the GRU RNN**

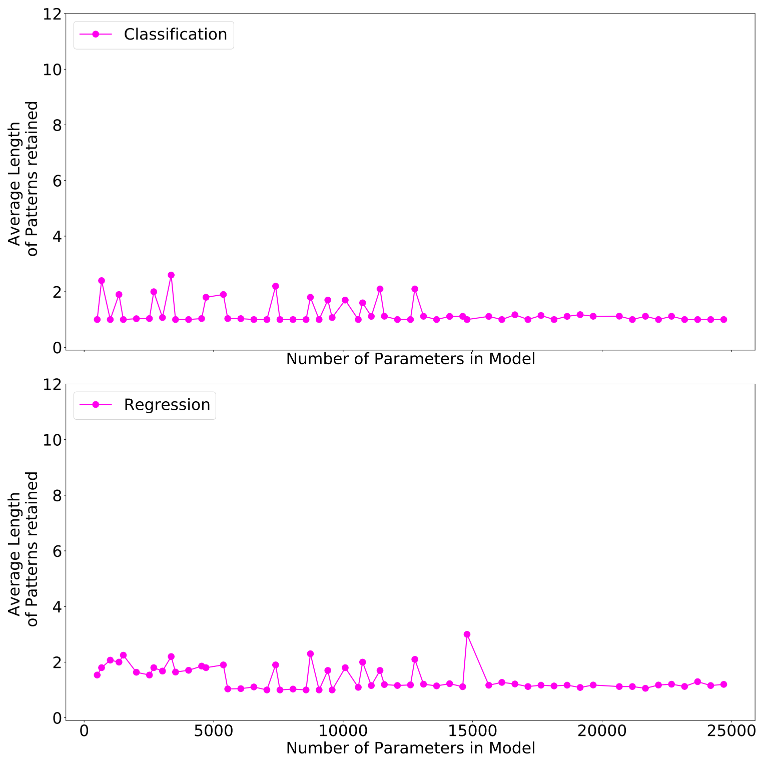

FIGURE 7.7: **Effect of the number of parameters on the ability to retain patterns for the Jordan RNN**

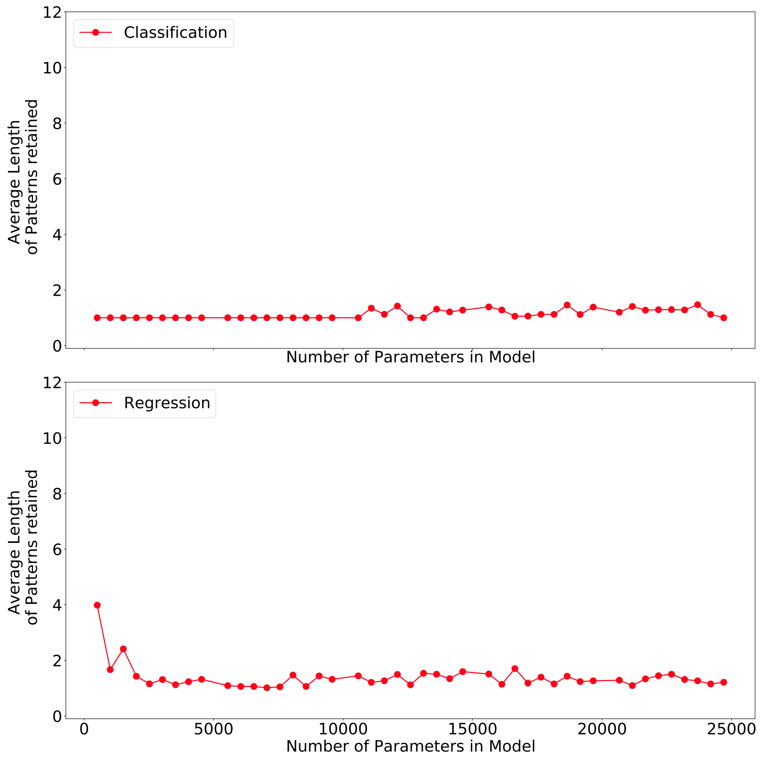

FIGURE 7.8: **Effect of the number of parameters on the ability to retain patterns for the LSTM RNN**

From figures 7.1, 7.2, 7.3, 7.4, 7.5, 7.6, 7.7 and 7.8 it appears that for smaller networks, regression formulation is able to retain longer sequences. In fact, for all networks, outside of the bidirectional Elman and bidirectional Jordan, the regression formulated RNNs retained longer sequences than the classification formulated counterparts. However, as observed previously, the regression formulated RNNs sequencer retention decays more as the number of parameters increased, whereas more parameters are more advantageous to the classification formulated counterparts.

Now that the classification- and retention formulated RNNs have been compared, the next investigation will focus on the RNNs themselves. The bidirectional GRU and bidirectional LSTM are observed to have the least decay in retention and thus can utilise an increase in the number of parameters more effectively than other RNNs before plateauing as illustrated in figures 7.2 and 7.4. For all RNNs excluding the Jordan RNN, regardless of the formulation of the problem statement, adding a bidirectional component increased the sequence retention ability of the RNN.

RNN	Classification	Regression
Bidirectional Elman	0.08614	-0.16808
Bidirectional GRU	0.42559	-0.26096
Bidirectional Jordan	-0.02675	-0.16129
Bidirectional LSTM	0.28802	-0.15808
Elman	0.12050	-0.15766
GRU	0.39138	-0.08353
Jordan	0.01249	-0.14990
LSTM	0.36145	-0.09158

TABLE 7.1: Spearman correlation between the number of patterns retained and the number of network parameters

To verify the relationship between the number of patterns retained and the number of network parameters, a further evaluation of the Spearman coefficient is presented in table 7.1. Here, from the coefficient presented, it is clear that all regression formulated RNN's sequence retention is negatively correlated to the number of parameters in the model, whereas most classification formulated RNN's are positively correlated. The RNN which gains the most benefit of increasing the number of parameters is the bidirectional GRU for the classification formulation, with a Spearman correlation coefficient of 0.42559. Interestingly the network which has the most negative correlation to the number of parameters is the bidirectional GRU for the regression formulation. Furthermore, it is clear that the bidirectional GRU is the most sensitive formulation choice with the largest correlation change. This confirms behavioural consistency in the claims made in this dissertation.

To confirm how effective each RNN type is at utilising the number of parameters given to it, the per parameter sequence capacity ratio is presented in figures 7.9, 7.10, 7.11, 7.12, 7.13, 7.14, 7.15 and 7.16. It is clear that in all cases, the per parameter sequence capacity decays as more parameters are added to the RNNs as observed previously. Initially the degradation is rapid, but as the more parameters are added, the more this ratio approaches zero, causing the illusion that the degradation in per parameter sequence capacity plateaus. Generally, RNNs which are more effective would be able to utilise more parameters to retain longer sequences. RNNs which are not as effective, may not be able to retain as much information as the number of parameters increase or may focus on a subset of patterns, a local optima.

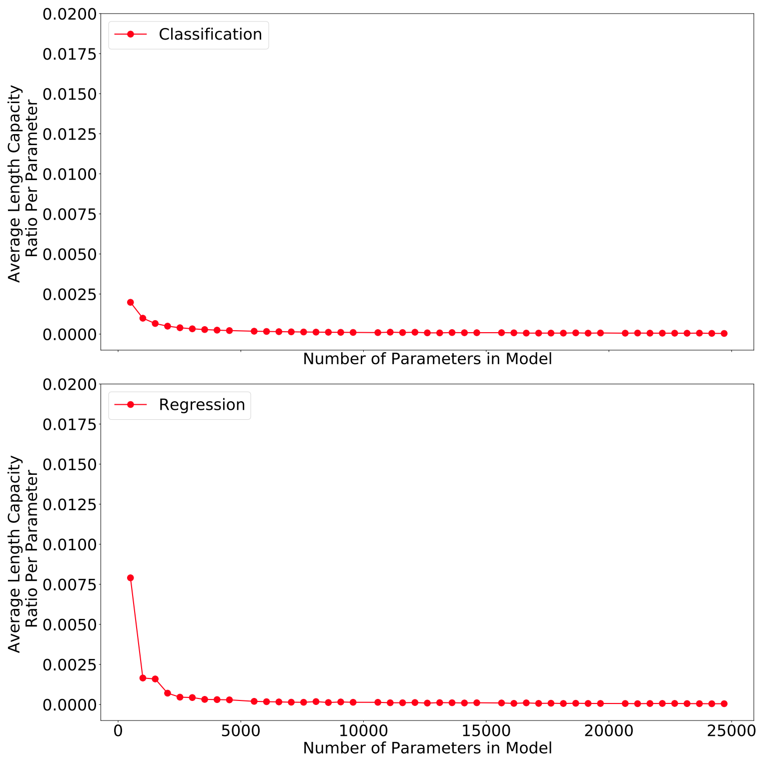

FIGURE 7.9: **Effect of increasing the number of parameters on the per parameter capacity of the LSTM RNN**

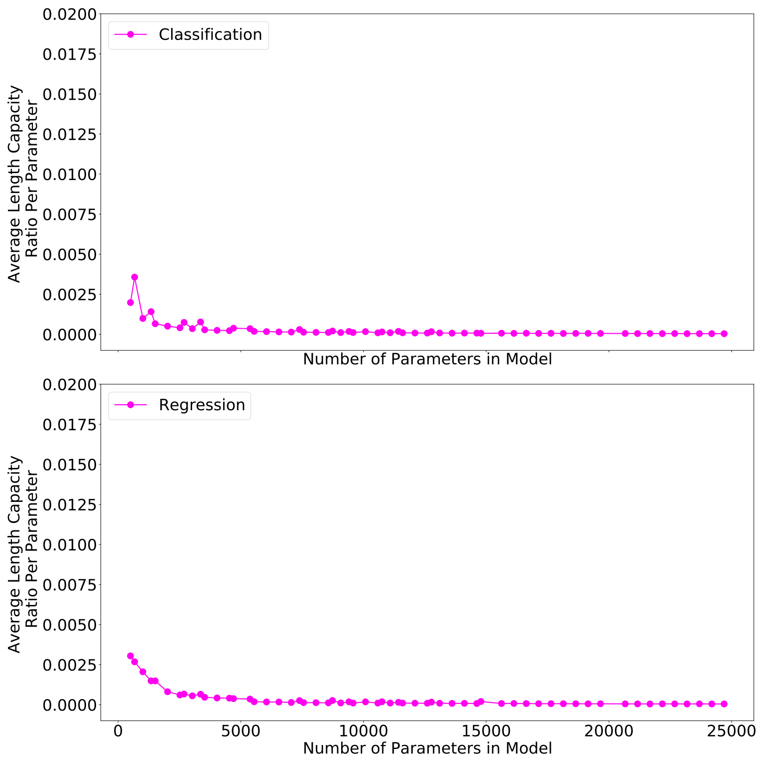

FIGURE 7.10: **Effect of increasing the number of parameters on the per parameter capacity of the Jordan RNN**

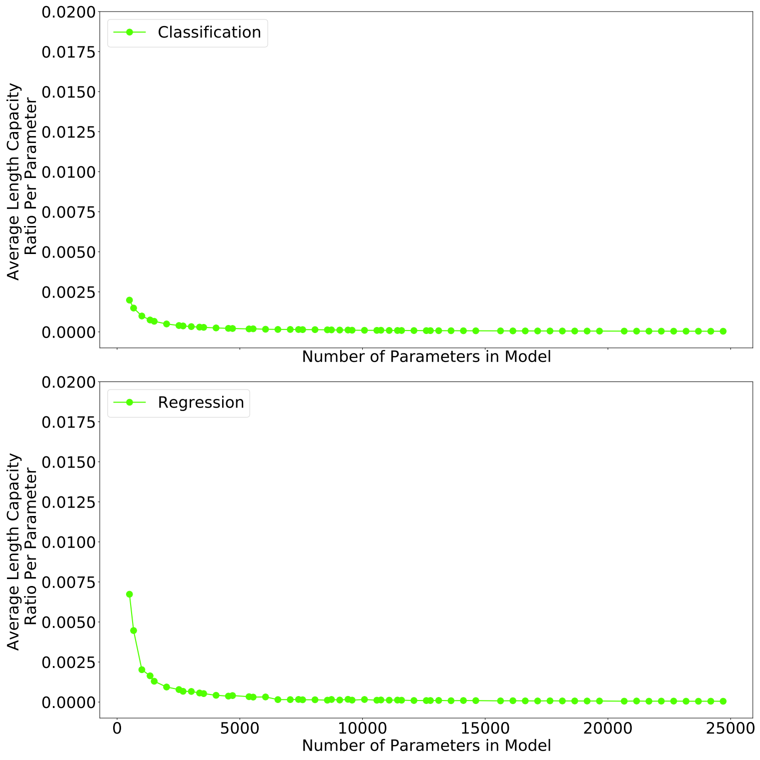

FIGURE 7.11: **Effect of increasing the number of parameters on the per parameter capacity of the bidirectional Jordan RNN**

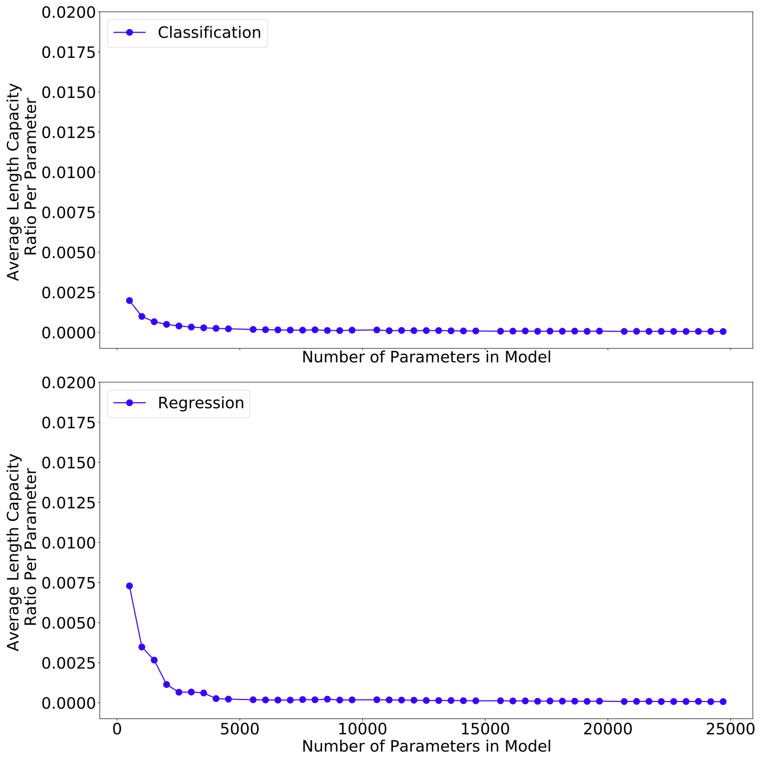

FIGURE 7.12: **Effect of increasing the number of parameters on the per parameter capacity of the GRU RNN**

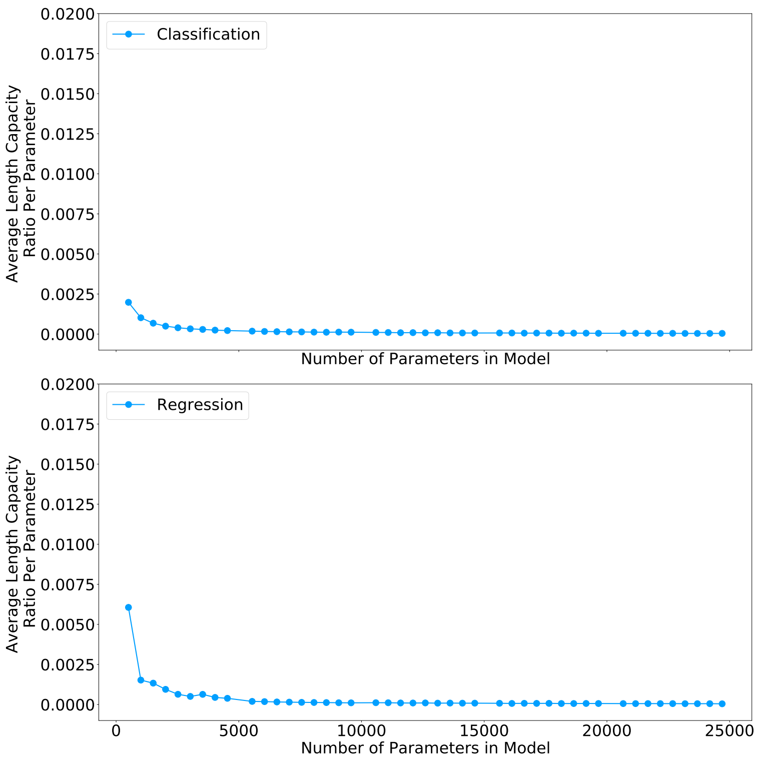

FIGURE 7.13: **Effect of increasing the number of parameters on the per parameter capacity of the Elman RNN**

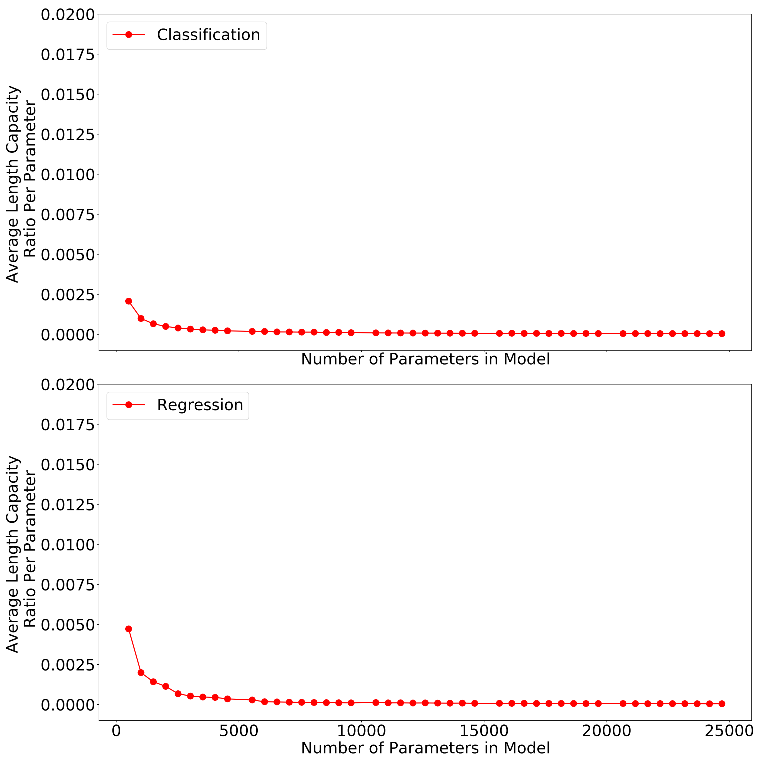

FIGURE 7.14: **Effect of increasing the number of parameters on the per parameter capacity of the bidirectional Elman RNN**

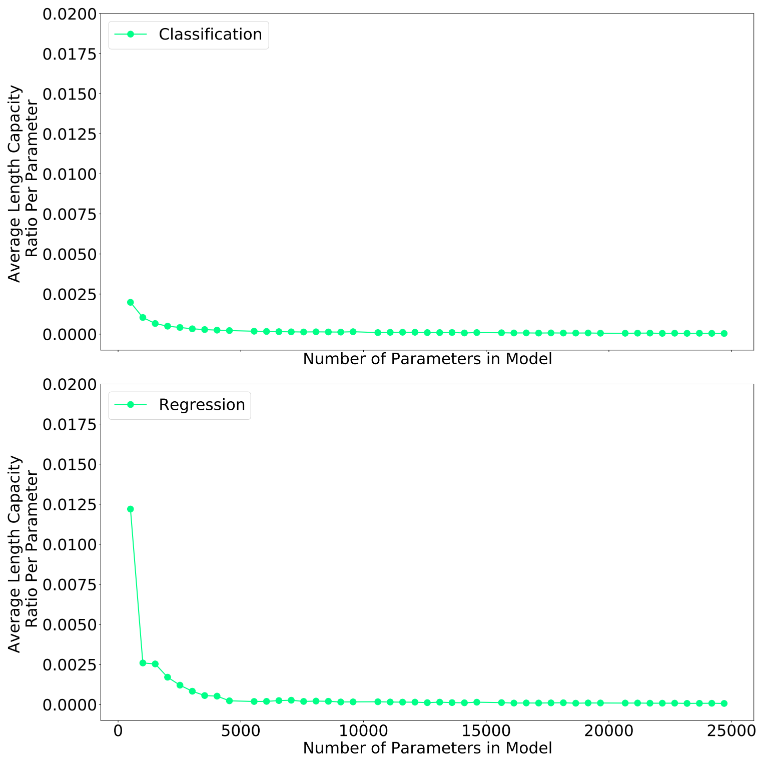

FIGURE 7.15: **Effect of increasing the number of parameters on the per parameter capacity of the bidirectional LSTM RNN**

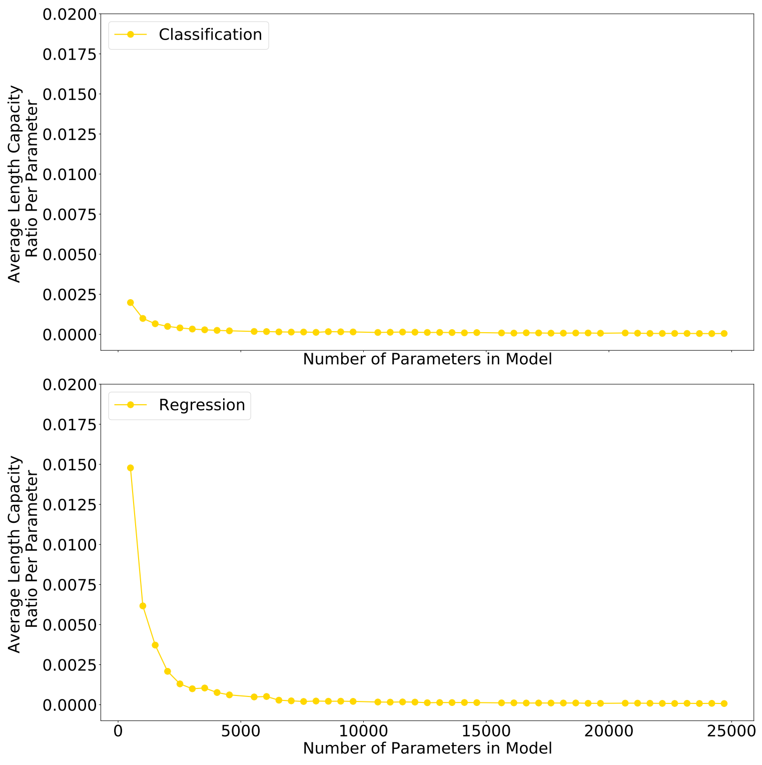

FIGURE 7.16: **Effect of increasing the number of parameters on the per parameter capacity of the bidirectional GRU RNN**

Further inspecting the per parameter sequence capacity ratios, as presented in table 7.2, there is a clear advantage in using the regression formulation for this problem statement. In all cases the RNNs achieve higher per parameter sequence capacity ratios in the regression formulated experiments when compared to the classification formulated experiments. This may have to do with the very nature of classification problems which require more output nodes. Upon further inspection, there is a clear advantage in using bidirectional layers, as the effectiveness of each parameter increases for the most part. Unsurprisingly, the RNN which had the best sequence capacity ratio is the bidirectional GRU, making it the most parameter efficient RNN in retaining longer sequences. This is unsurprising as the bidirectional GRU has the highest Spearman Rank coefficient as presented in table 7.1.

RNN	Classification	Regression
Bidirectional Elman	0.00011	0.00014
Bidirectional GRU	0.00014	0.00030
Bidirectional Jordan	0.00011	0.00016
Bidirectional LSTM	0.00012	0.00023
Elman	0.00012	0.00015
GRU	0.00013	0.00021
Jordan	0.00012	0.00014
LSTM	0.00012	0.00016

Table 7.2: Average Number of patterns retained per single network parameter over all network configurations

7.1.2 Average Per Parameter Sequence Length Retention for each Activation Function

In this section, the effect that the activation choice has on RNN sequence length retention is investigated.

The average sequence length that was retained by each activation function over increasing number of parameters are shown in figures 7.17, 7.18, 7.19, 7.20, 7.21, 7.22, 7.23, 7.24, 7.25 and 7.26. For all activation functions, the average pattern length retained decays as the parameter count increases. From the aforementioned figures it can be seen that the elu activation function resulted in the longest sequence retained at 6.3 patterns on average. However the network loses this ability as the number of parameters increase as observed in figure 7.20. The relu activation function had a better retention ratio as the number of parameters increased when compared to the effect of other activation functions on average. If a comparison is made between the classification and regression cases, it can clearly be determined that all activation functions favour the regression problem statement for this experiment. Similar results were obtained in section 7.1.1. The same per parameter degradation can be seen as observed in the previous section 7.1.1 with the most effected activation functions being elu, linear, relu, selu, sigmoid and tanh. The least affected being: softmax, softplus and softsign. The activation functions least affected could be due to the range and form of these functions. Although elu is similar to softplus, the slope of the softplus is slightly more gradual than the former which might be advantageous.

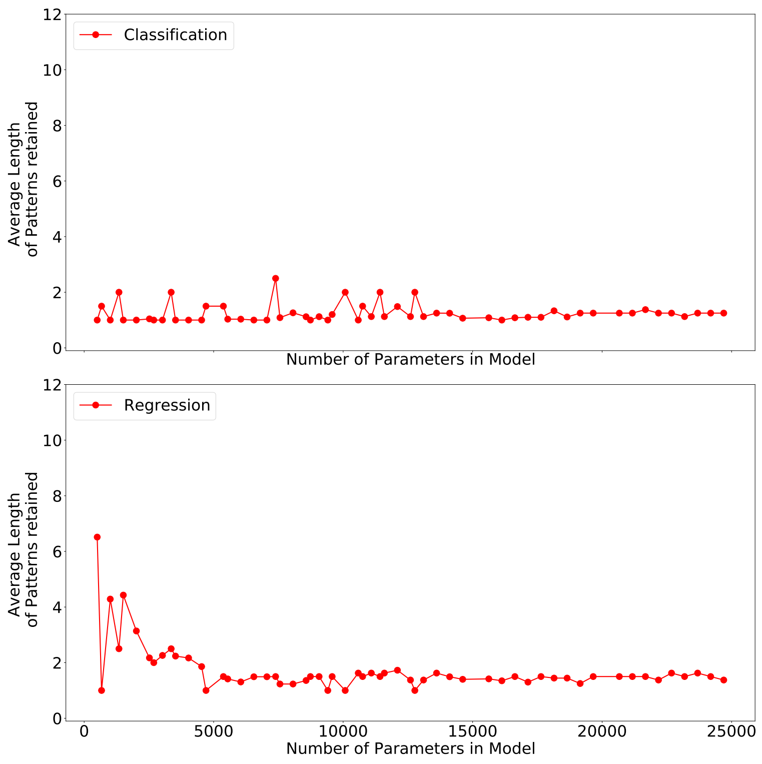

FIGURE 7.17: **Effect of the number of parameters on the ability to retain patterns over each activation function: elu**

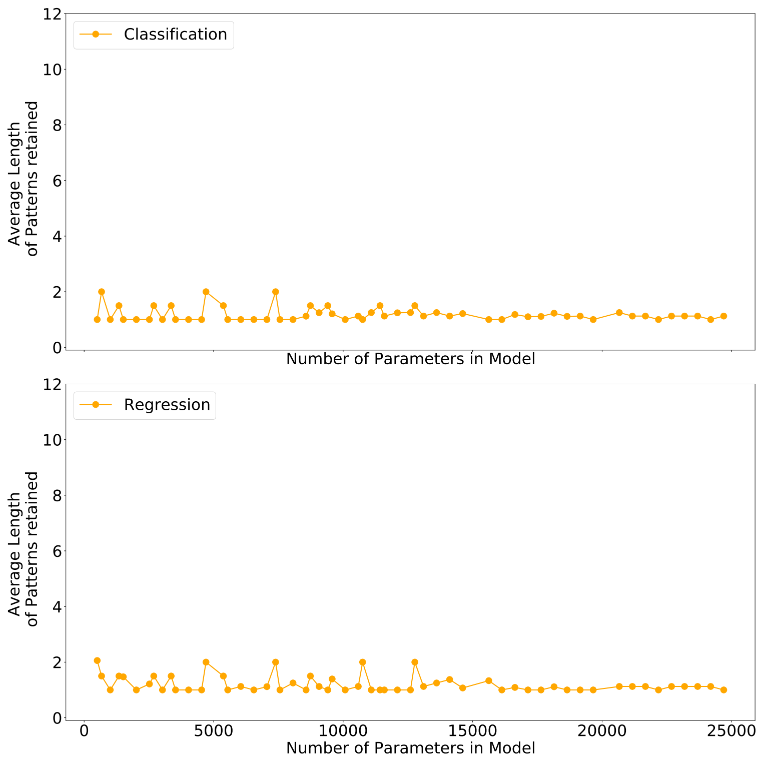

FIGURE 7.18: **Effect of the number of parameters on the ability to retain patterns over each activation function: hard sigmoid**

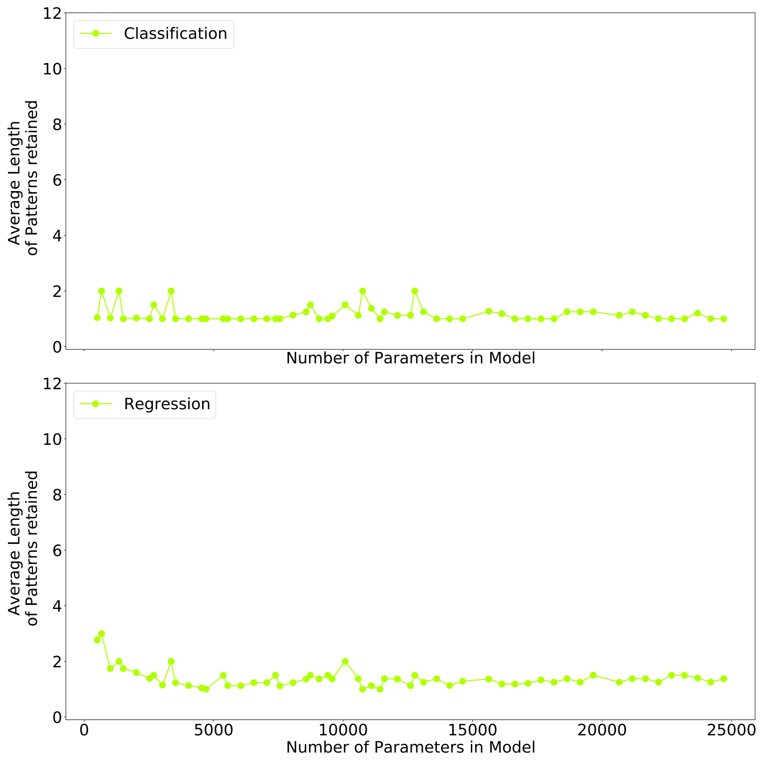

FIGURE 7.19: **Effect of the number of parameters on the ability to retain patterns over each activation function: linear**

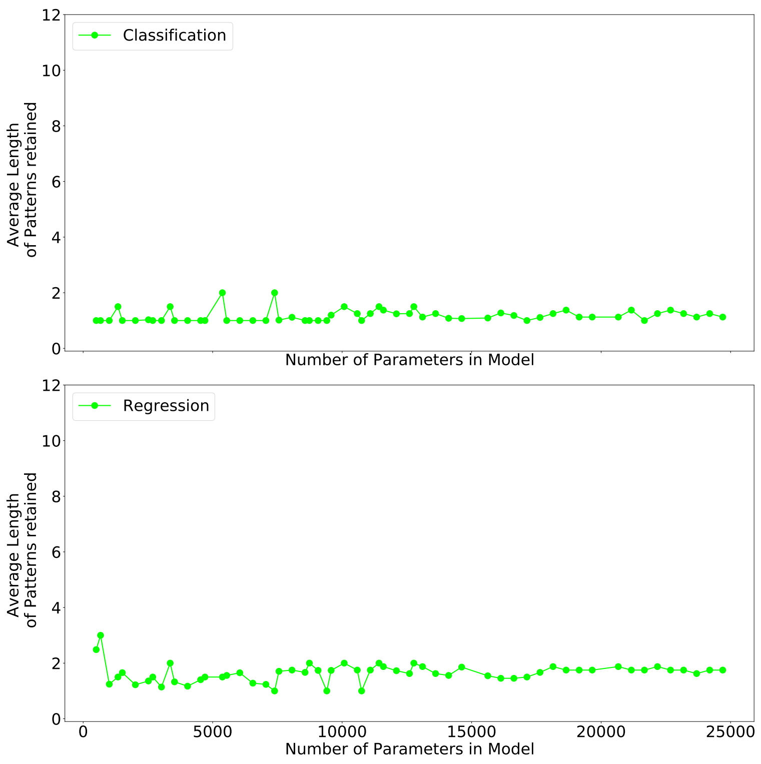

FIGURE 7.20: **Effect of the number of parameters on the ability to retain patterns over each activation function: relu**

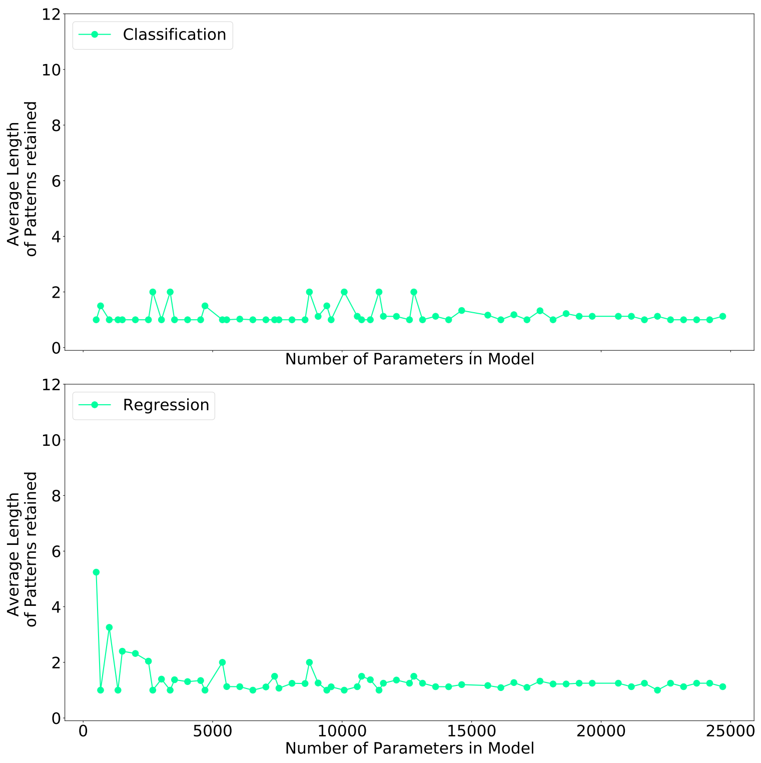

FIGURE 7.21: **Effect of the number of parameters on the ability to retain patterns over each activation function: selu**

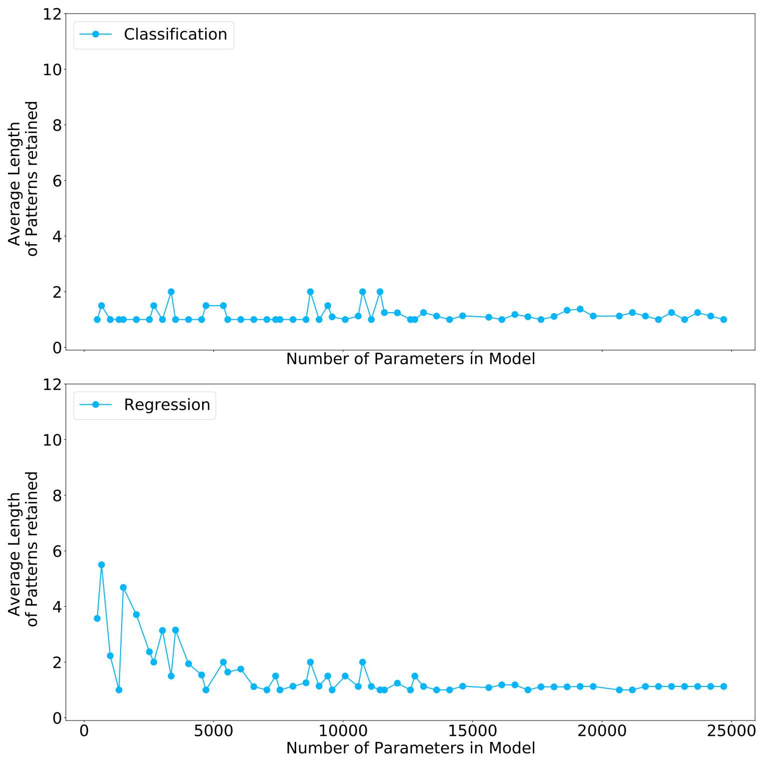

FIGURE 7.22: **Effect of the number of parameters on the ability to retain patterns over each activation function: sigmoid**

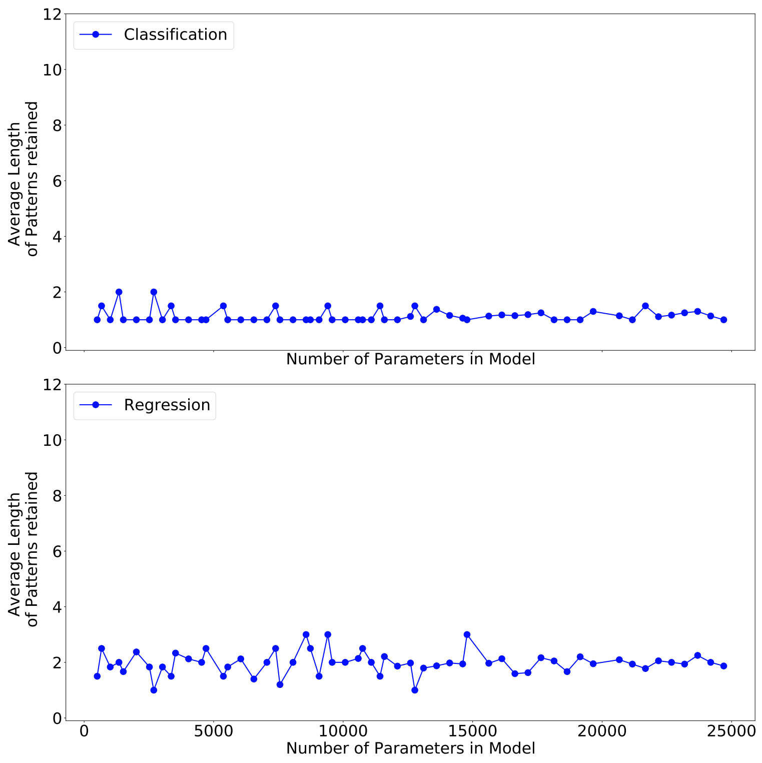

FIGURE 7.23: **Effect of the number of parameters on the ability to retain patterns over each activation function: softmax**

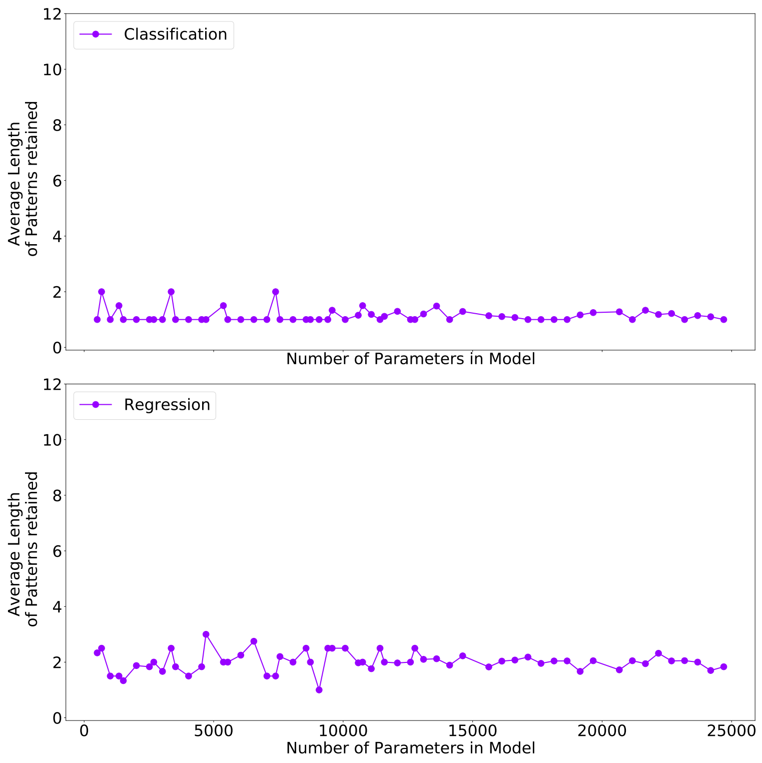

FIGURE 7.24: **Effect of the number of parameters on the ability to retain patterns over each activation function: softplus**

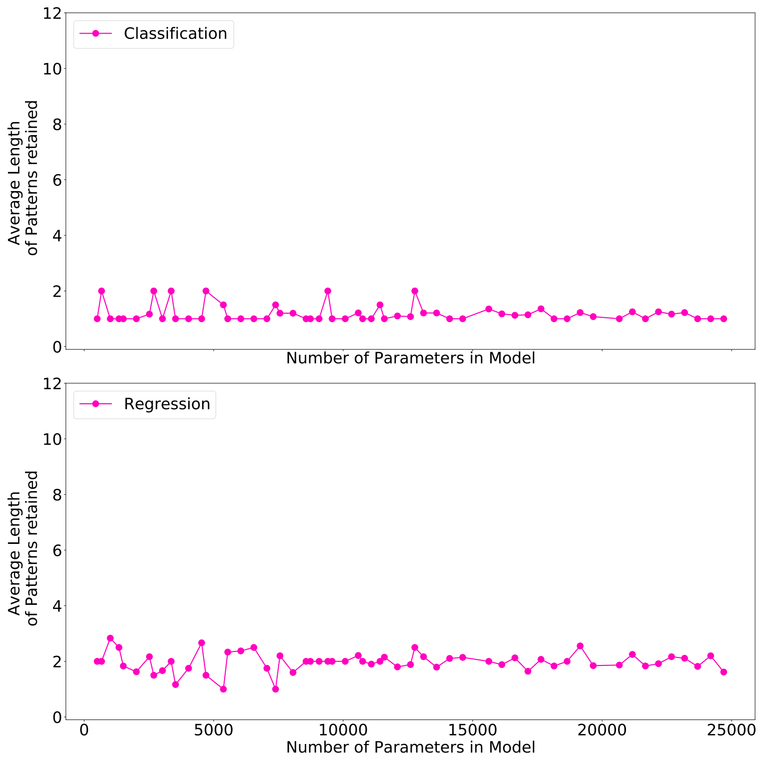

FIGURE 7.25: **Effect of the number of parameters on the ability to retain patterns over each activation function: softsign**

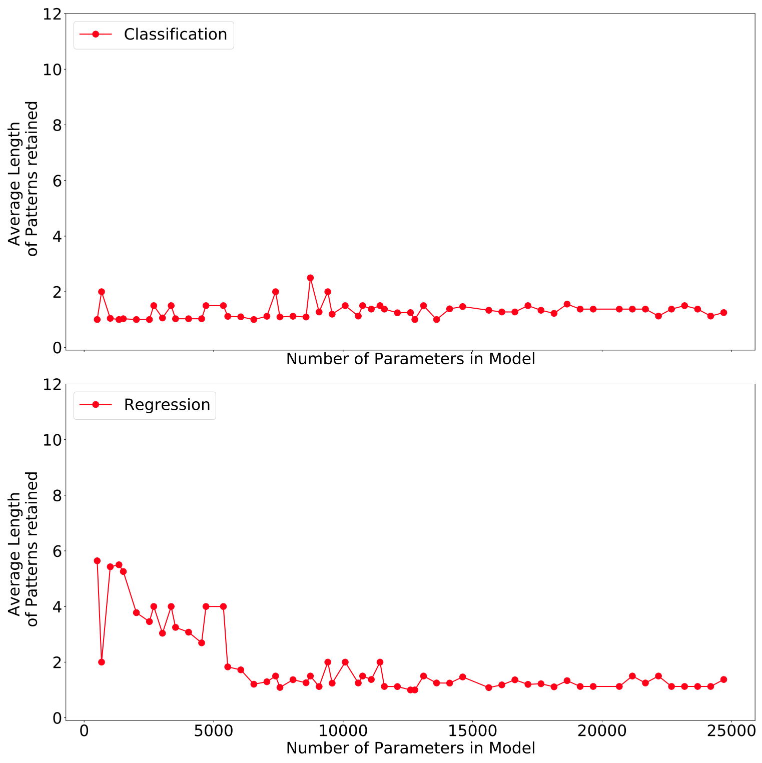

FIGURE 7.26: **Effect of the number of parameters on the ability to retain patterns over each activation function: tanh**

To determine which activation function is affected the worst by an increase in the number of parameters, the Spearman Rank correlation, as defined in section 6.1.2, between the length of the pattern retained and the number of parameters is presented in table 7.3 and 7.4. As shown in tables 7.3 and 7.4 the most negatively correlated activation functions are the elu and tanh with coefficients -0.2842 and -0.2997 in the regression case respectively, indicating that the performance of networks using these activation functions will be most negatively affected when increasing the number of parameters. For classification, the softplus activation function using the Jordan RNN had the most negatively correlated relationship between activation function used and per parameter capacity with coefficient -0.4182. On the opposite side, the most correlated increase in sequence retention as the number of parameters grew, is the relu activation function applied to bidirectional GRU during the classification problem statement with coefficient 0.7625. Thus, this configuration should have the best overall sequence retention when applied to classification formulation of this experiment. Since the behavioural traits between the per parameter capacity and activation function used have been explored, the next investigation will determine how the activation functions affect the performance length capacity ratio of each network.

Figures 7.27, 7.28, 7.29, 7.30, 7.31, 7.32, 7.33, 7.34, 7.35 and 7.36 capture the per parameter sequence retention capacity for an increasing number of parameters for each RNN type under each considered activation function. In general the behavioural traits for each RNN type under each considered activation function remain the same as observed in section 7.1.1.

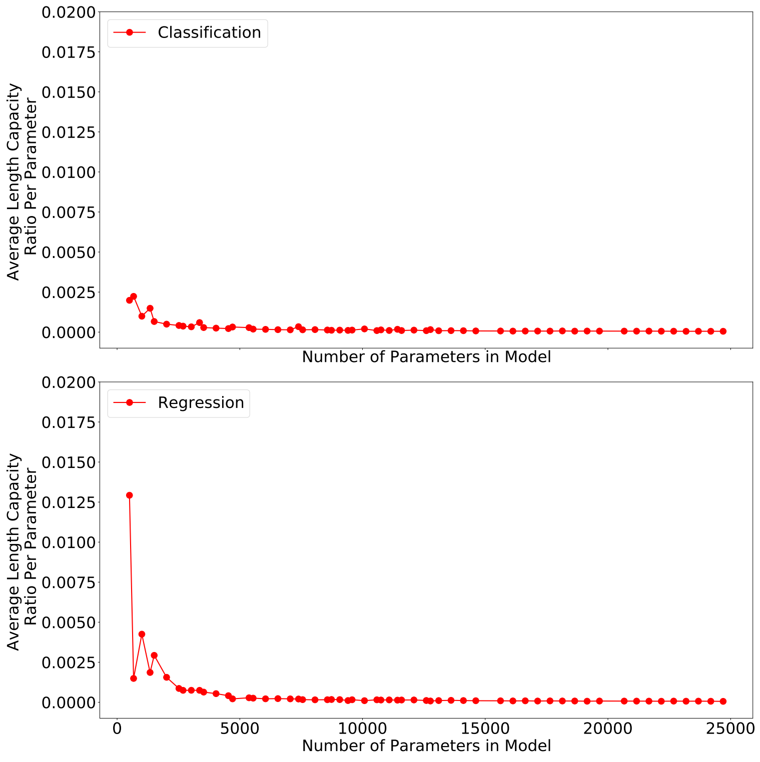

FIGURE 7.27: **Effect of increasing the number of parameters on the per parameter capacity of the network using the elu activation function**

Activation Function		Bidirectional Elman	Bidirectional GRU	Bidirectional Jordan	Bidirectional LSTM	Elman	GRU	Jordan	LSTM
elu	Classification	-0.0384	0.6711	-0.0431	0.3913	0.0000	0.4049	-0.0358	0.4175
	Regression	-0.4271	-0.3294	-0.3847	-0.1332	-0.3025	-0.2610	-0.3081	-0.1289
hard sigmoid	Classification	0.0000	0.2820	0.0000	0.3145	0.0000	0.3851	-0.0668	0.3797
	Regression	0.0000	-0.1824	-0.1283	-0.1481	0.0000	0.2006	-0.0561	0.2993
linear	Classification	-0.1755	0.3435	0.0000	0.2628	-0.1065	0.2490	-0.0840	0.3446
	Regression	-0.2164	-0.3602	-0.2194	0.4085	-0.1769	0.1586	-0.1934	0.3453
relu	Classification	0.0000	0.7625	0.0000	0.2263	0.0000	0.4545	-0.0678	0.3525
	Regression	-0.2260	-0.0600	-0.2162	0.0955	-0.1731	0.5743	-0.0788	0.0891
selu	Classification	0.0000	0.2962	0.0000	0.2240	0.0000	0.3482	-0.0252	0.3486
	Regression	0.0000	-0.3322	0.0000	-0.2840	-0.1570	-0.3069	-0.0122	-0.3273
sigmoid	Classification	0.0000	0.4493	0.0000	0.3477	0.0000	0.4713	-0.0344	0.2854
	Regression	-0.2084	-0.3223	-0.1529	-0.2653	-0.1576	-0.2330	-0.0475	-0.1304

Table 7.3: Spearman correlation between number of patterns retained and the number of network parameters per activation function for each RNN type

Activation Function		Bidirectional Elman	Bidirectional GRU	Bidirectional Jordan	Bidirectional LSTM	Elman	GRU	Jordan	LSTM
softmax	Classification	0.0000	0.2474	0.0000	0.2500	0.0000	0.4533	-0.3428	0.3297
	Regression	0.1914	0.0680	-0.1159	-0.0105	0.0607	-0.1138	-0.0979	0.0061
softplus	Classification	0.0000	0.1868	0.0000	0.1654	0.0000	0.1720	-0.4182	0.2968
	Regression	-0.0334	0.1132	-0.1809	0.0414	0.1808	0.0049	-0.0221	0.0618
softsign	Classification	0.0000	0.0679	0.0000	0.1406	0.0000	0.1240	-0.3432	0.4664
	Regression	0.0666	0.0589	-0.0135	-0.2120	0.0459	-0.0342	0.2149	-0.1822
tanh	Classification	0.3786	0.3551	-0.0329	0.3121	0.4548	0.5208	0.3310	0.3282
	Regression	-0.4351	-0.2762	-0.3937	-0.2881	-0.4194	-0.0516	-0.4278	-0.1054

TABLE 7.4: Spearman correlation between the number of patterns retained and the number of network parameters per activation function for each RNN type

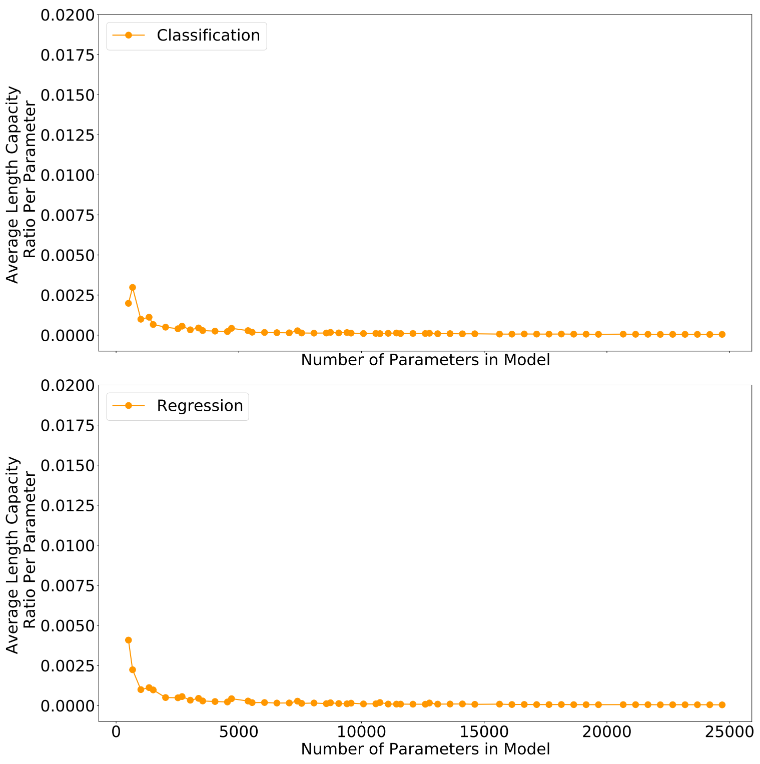

FIGURE 7.28: **Effect of increasing the number of parameters on the per parameter capacity of the network using the hard sigmoid activation function**

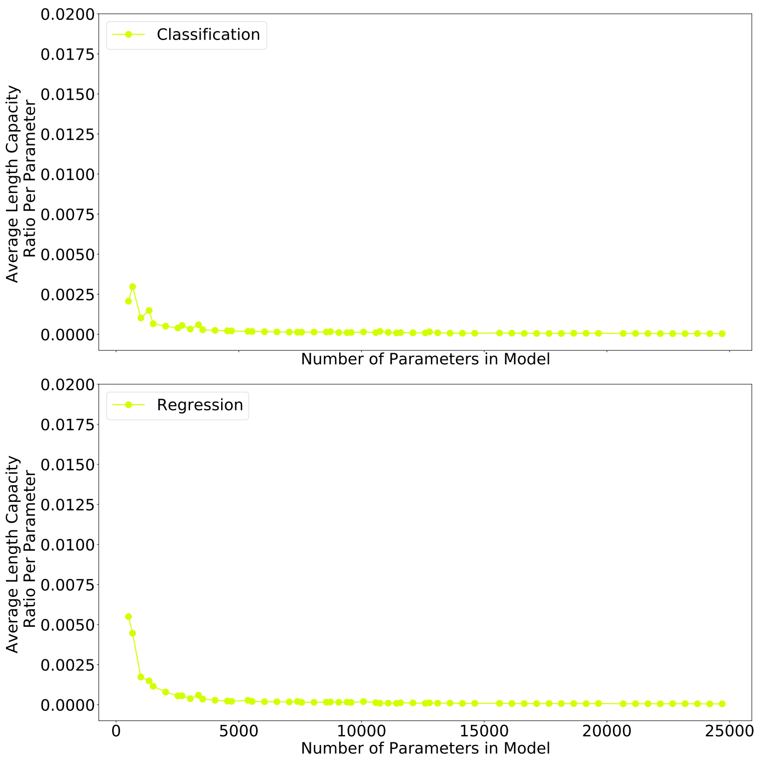

FIGURE 7.29: **Effect of increasing the number of parameters on the per parameter capacity of the network using the linear activation function**

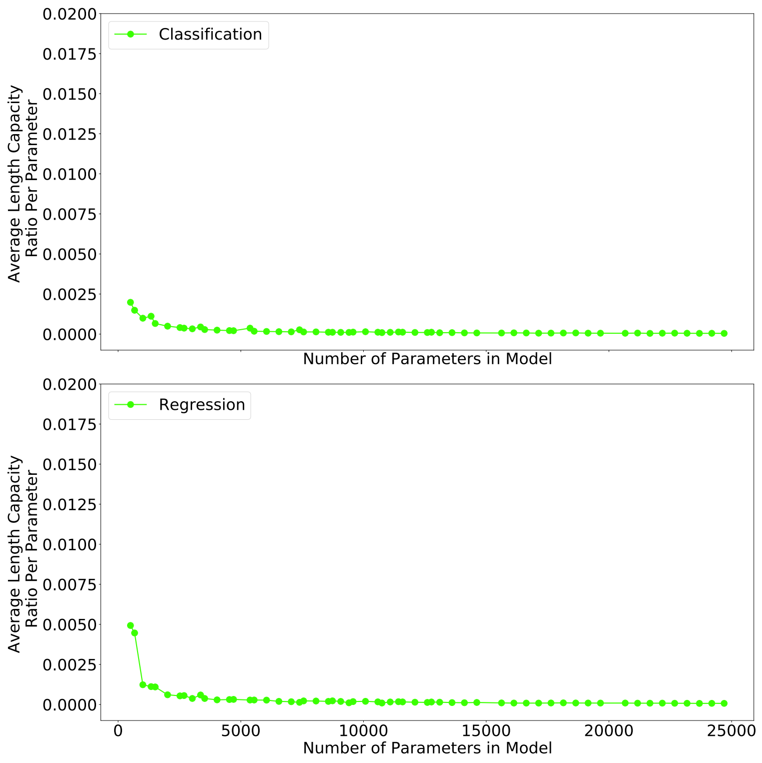

FIGURE 7.30: **Effect of increasing the number of parameters on the per parameter capacity of the network using the relu activation function**

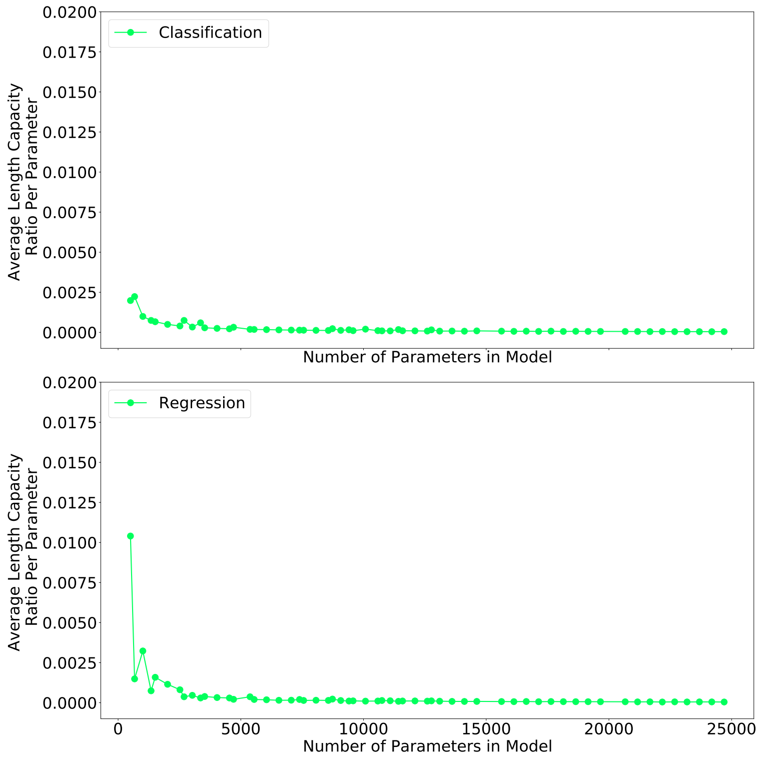

FIGURE 7.31: **Effect of increasing the number of parameters on the per parameter capacity of the network using the selu activation function**

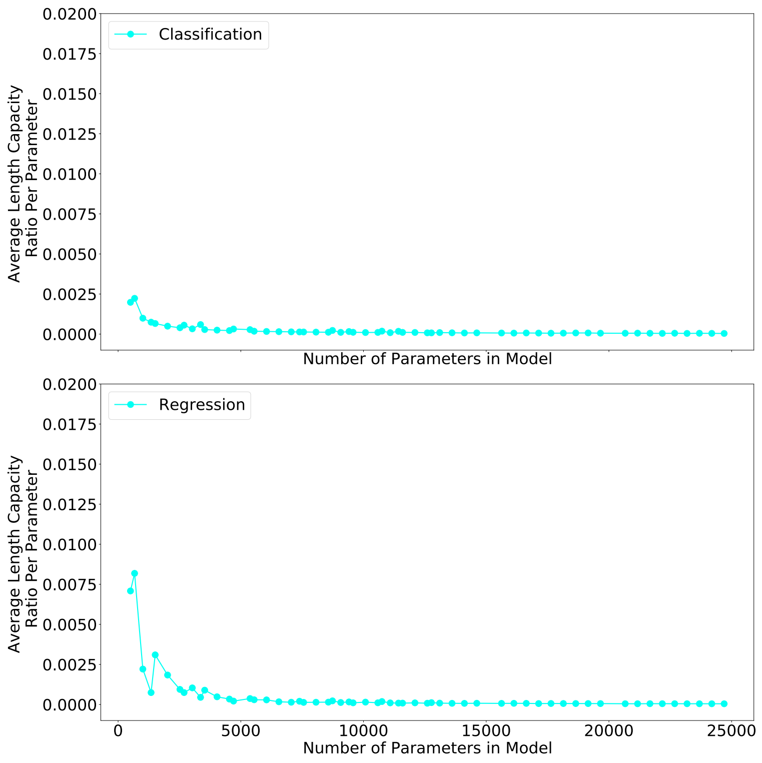

Figure 7.32: **Effect of increasing the number of parameters on the per parameter capacity of the network using the sigmoid activation function**

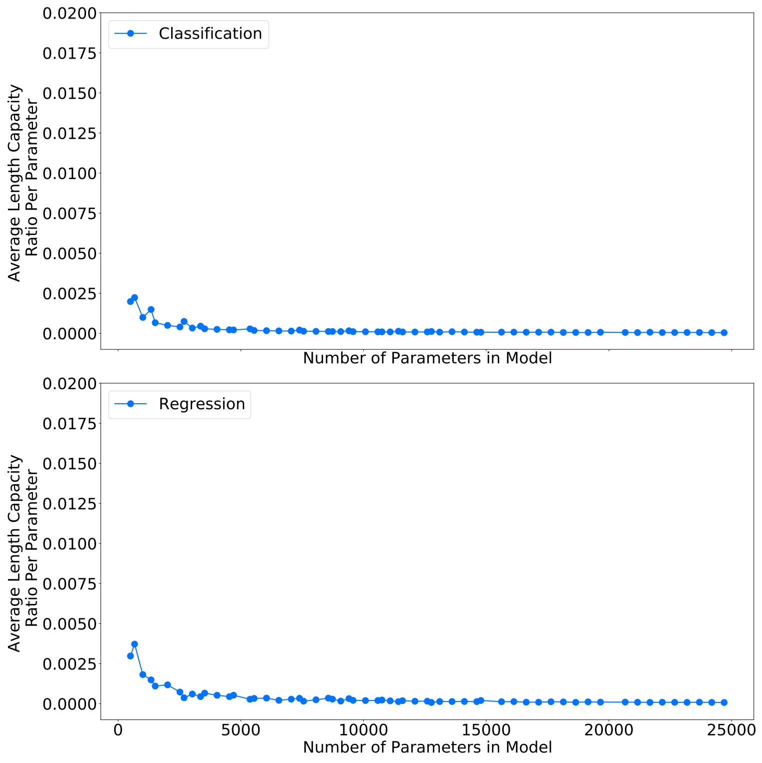

Figure 7.33: **Effect of increasing the number of parameters on the per parameter capacity of the network using the softmax activation function**

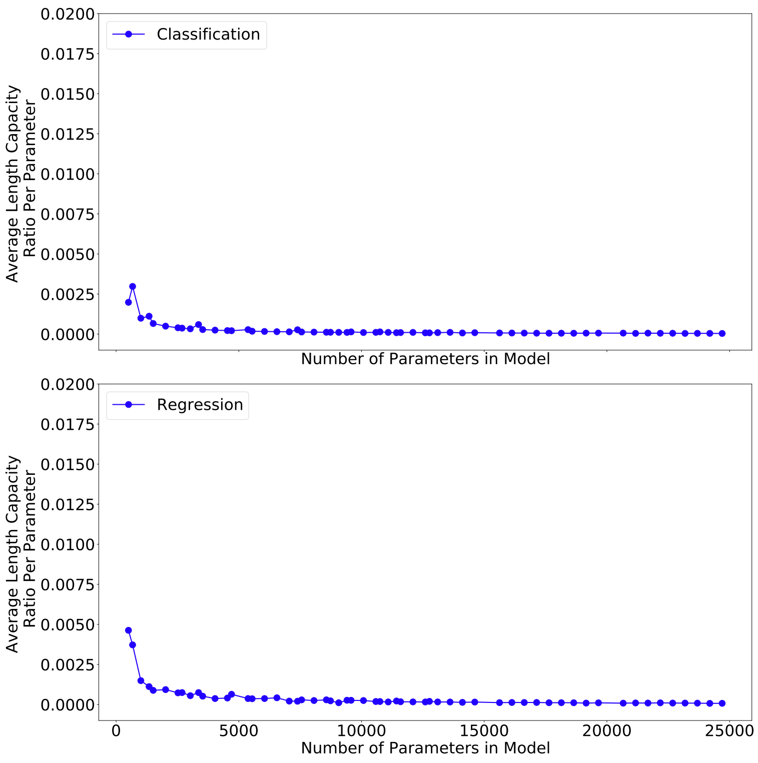

FIGURE 7.34: **Effect of increasing the number of parameters on the per parameter capacity of the network using the softplus activation function**

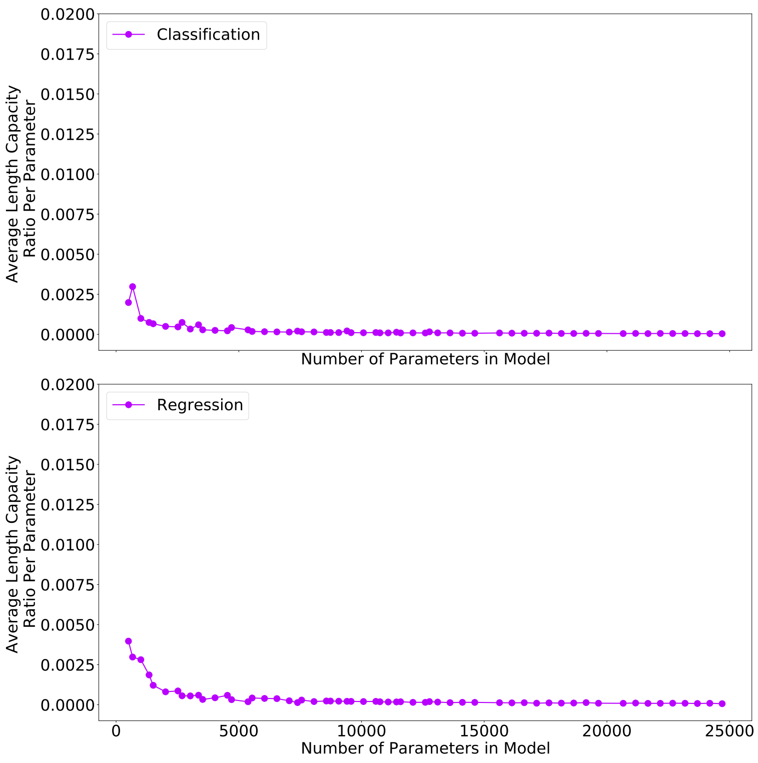

FIGURE 7.35: **Effect of increasing the number of parameters on the per parameter capacity of the network using the softsign activation function**

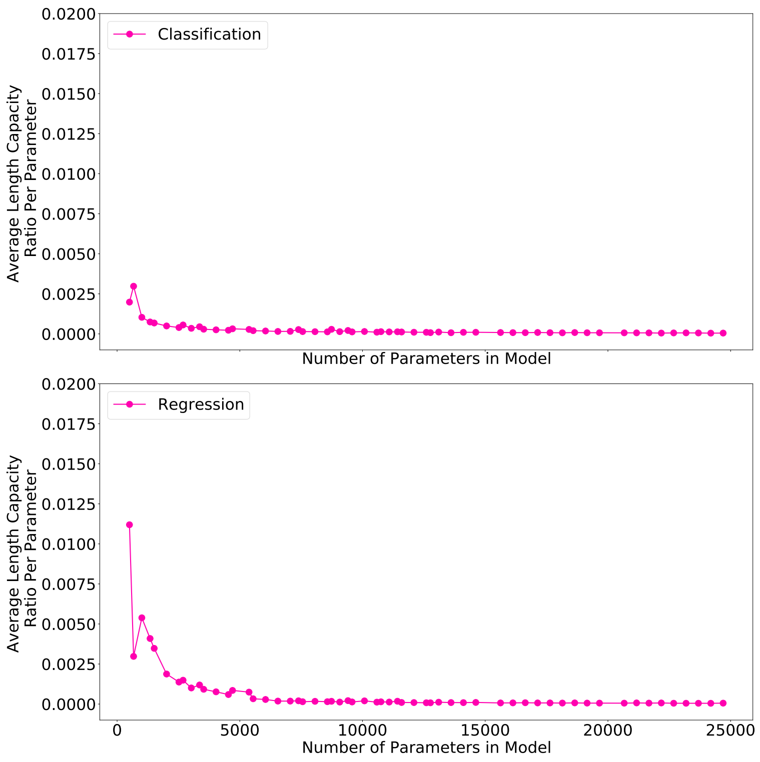

FIGURE 7.36: **Effect of increasing the number of parameters on the per parameter capacity of the network using the tanh activation function**

Table 7.5 captures the per parameter sequence capacity ratio for each RNN using different activation functions in both the regression and classification formulation of this experiment. In the case of regression, the best pairing of elu activation function and RNN type is the bidirectional GRU using the elu activation function at 0.00047 patterns per parameter. Followed by the same configuration, but using the sigmoid activation function which resulted in 0.00038 patterns per parameter. For the classification formulation the best configuration is still the bidirectional GRU paired with the elu activation function, which resulted in 0.00016 patterns per parameter retained. However, the tanh activation function achieved the highest average per parameter sequence capacity over all RNNs in both the regression and classification formulation with an average of 0.00026 and 0.00014 patterns per parameter retained respectively.

Activation Function		bidirectional Elman	bidirectional GRU	bidirectional Jordan	bidirectional LSTM	Elman	GRU	Jordan	LSTM
elu	Classification	0.00011	0.00016	0.00012	0.00013	0.00012	0.00014	0.00012	0.00013
	Regression	0.00015	0.00047	0.00016	0.00034	0.00023	0.00026	0.00018	0.00015
hard sigmoid	Classification	0.00011	0.00014	0.00012	0.00014	0.00012	0.00014	0.00012	0.00013
	Regression	0.00010	0.00015	0.00011	0.00017	0.00011	0.00013	0.00011	0.00013
linear	Classification	0.00012	0.00014	0.00012	0.00013	0.00012	0.00014	0.00012	0.00013
	Regression	0.00011	0.00028	0.00012	0.00023	0.00012	0.00014	0.00012	0.00013
relu	Classification	0.00012	0.00016	0.00012	0.00013	0.00012	0.00014	0.00012	0.00013
	Regression	0.00012	0.00033	0.00012	0.00013	0.00012	0.00035	0.00011	0.00013
selu	Classification	0.00011	0.00014	0.00012	0.00012	0.00012	0.00013	0.00012	0.00013
	Regression	0.00010	0.00033	0.00011	0.00022	0.00011	0.00023	0.00011	0.00026
sigmoid	Classification	0.00011	0.00014	0.00012	0.00012	0.00012	0.00014	0.00012	0.00013
	Regression	0.00011	0.00038	0.00015	0.00033	0.00011	0.00029	0.00011	0.00017
softmax	Classification	0.00007	0.00009	0.00007	0.00009	0.00008	0.00009	0.00008	0.00009
	Regression	0.00013	0.00015	0.00013	0.00013	0.00014	0.00014	0.00012	0.00014
softplus	Classification	0.00007	0.00009	0.00007	0.00009	0.00008	0.00009	0.00007	0.00010
	Regression	0.00013	0.00014	0.00014	0.00013	0.00014	0.00014	0.00013	0.00013
softsign	Classification	0.00008	0.00009	0.00007	0.00009	0.00008	0.00011	0.00009	0.00010
	Regression	0.00016	0.00015	0.00013	0.00014	0.00015	0.00015	0.00013	0.00015
tanh	Classification	0.00014	0.00013	0.00012	0.00013	0.00015	0.00014	0.00015	0.00013
	Regression	0.00028	0.00033	0.00036	0.00029	0.00026	0.00014	0.00025	0.00015

TABLE 7.5: Per parameter capacity between length of patterns retained and the number of network parameters per activation function for each RNN

7.1.3 Relationship between Sequence Length Retention and the Number Of Layers

The section investigates the impact that increasing the number of layers has on the ability of RNNs to retain sequences of increasing length. Here an average is taken across all configurations of RNNs using multiple different activation functions as presented in the previous section 7.1.2.

The findings of the effect of increasing the number of layers in RNNs has on the length of pattern retained, is illustrated in figures 7.37, 7.38, 7.39, 7.40, 7.41, 7.42, 7.43 and 7.44. For each network the average length of a pattern that was retained degraded drastically from using one hidden layer to two. Thereafter a slight rise in the average length of patterns retained was observed for the all RNNs except for the bidirectional GRU when using three hidden layers. Increasing the number of hidden layers did not yield much longer sequence retention. For finer granular analysis of behavioural traits for each network at different layer configurations, the Spearman correlations reported in table 7.6 is considered.

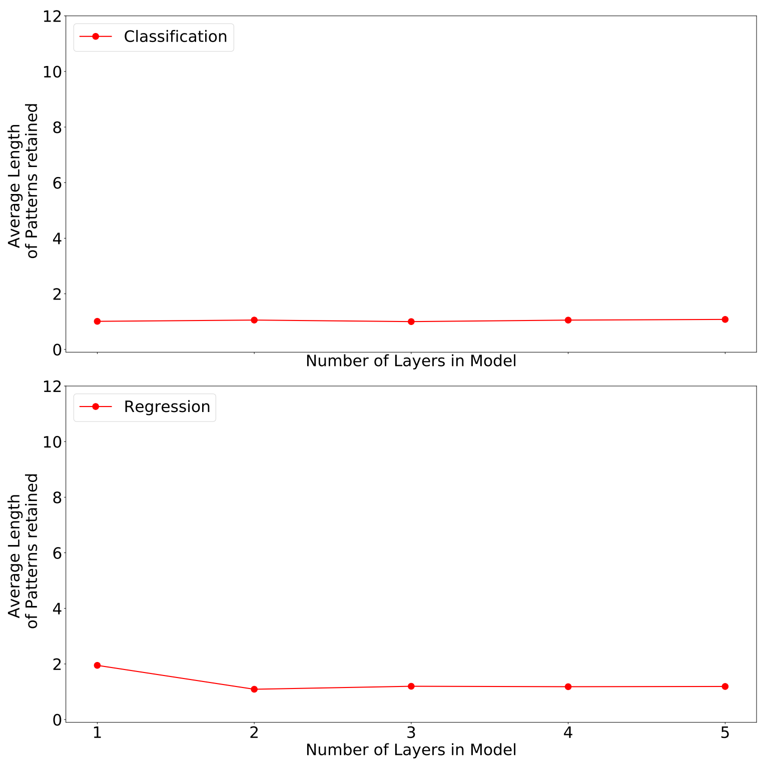

FIGURE 7.37: **Effect of increasing number of layers on the ability to retain patterns for the Bidirectional Elman RNN**

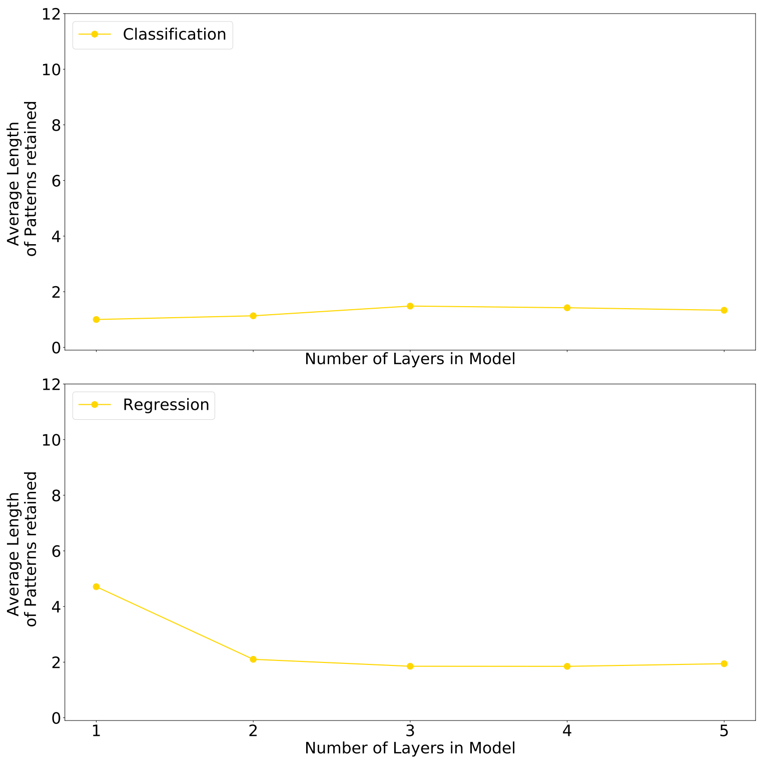

FIGURE 7.38: **Effect of increasing number of layers on the ability to retain patterns for the Bidirectional GRU RNN**

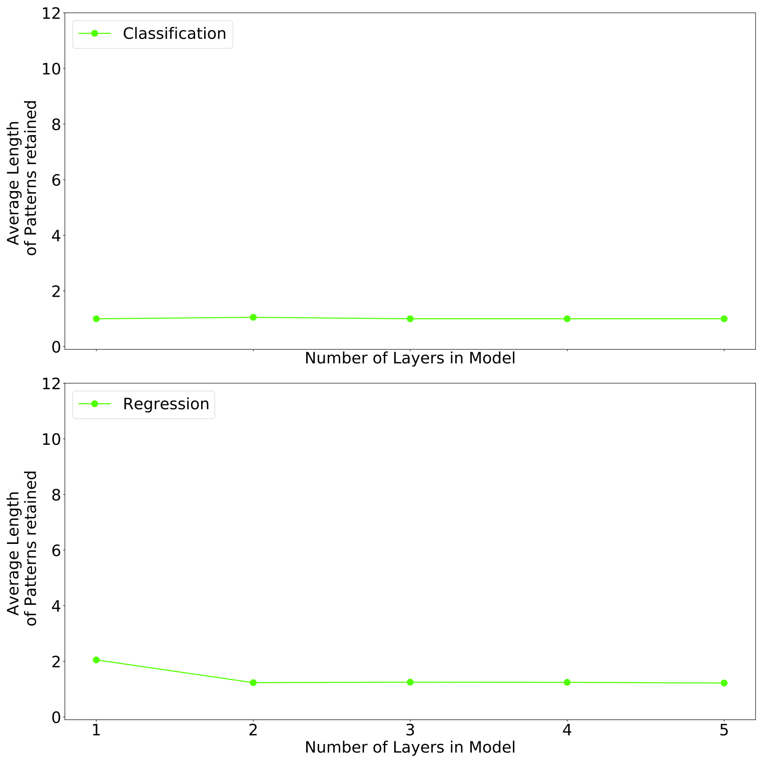

FIGURE 7.39: **Effect of increasing number of layers on the ability to retain patterns for the Bidirectional Jordan RNN**

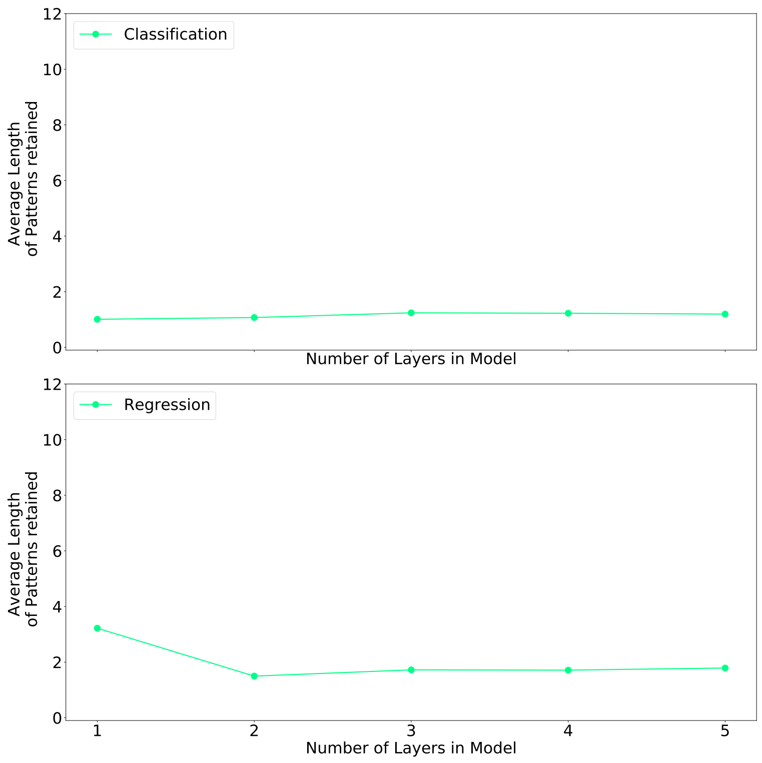

FIGURE 7.40: **Effect of increasing number of layers on the ability to retain patterns for the bidirectional LSTM RNN**

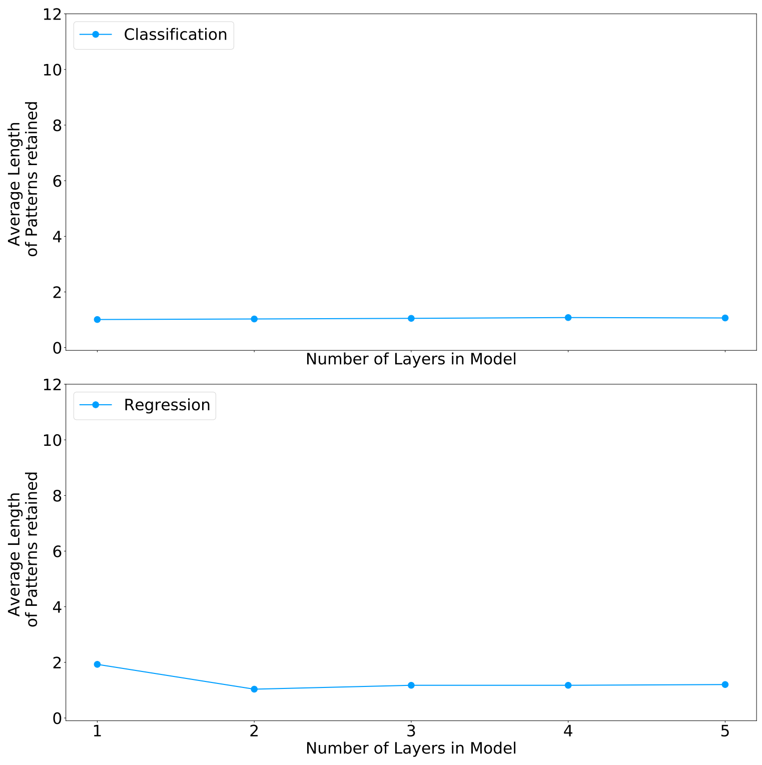

FIGURE 7.41: **Effect of increasing number of layers on the ability to retain patterns for the Elman RNN**

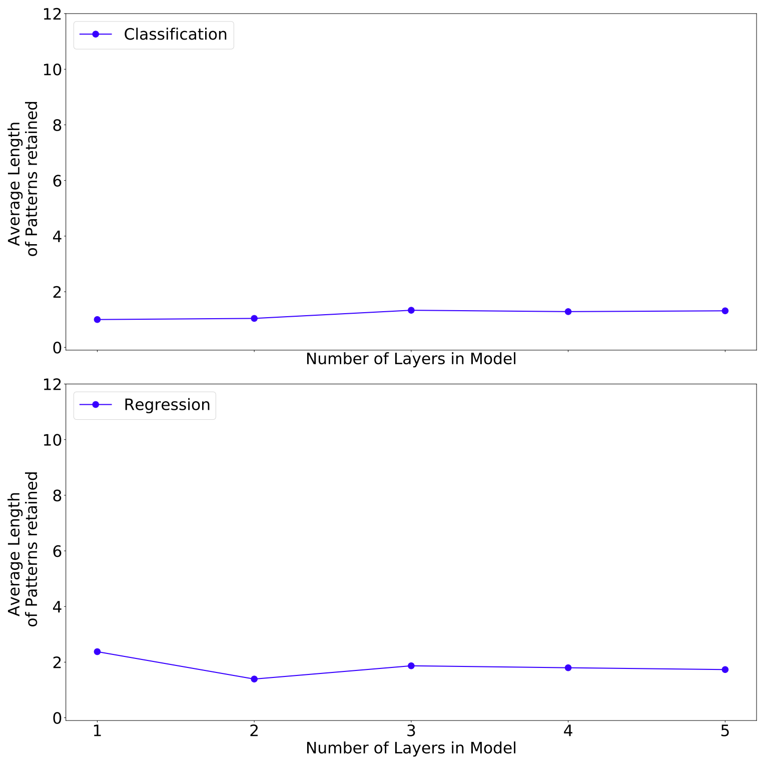

Figure 7.42: **Effect of increasing number of layers on the ability to retain patterns for the GRU RNN**

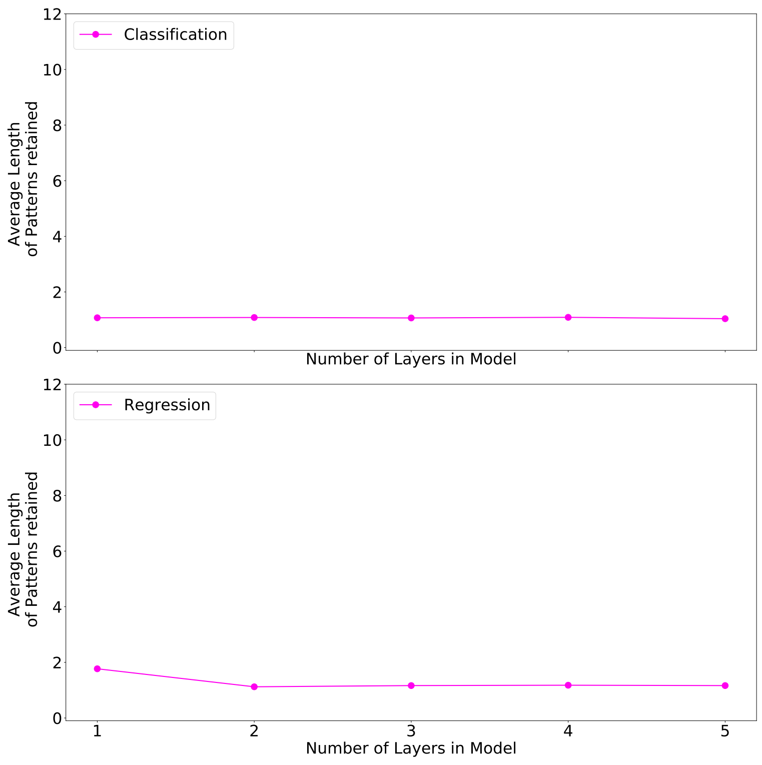

Figure 7.43: **Effect of increasing number of layers on the ability to retain patterns for the Jordan RNN**

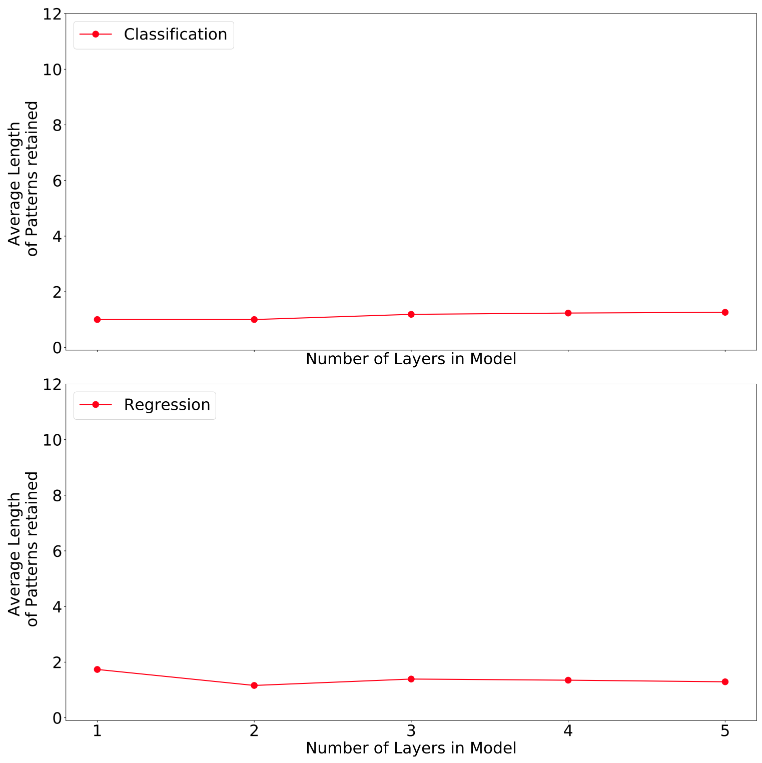

FIGURE 7.44: **Effect of increasing number of layers on the ability to retain patterns for the LSTM RNN**

In table 7.6 the Spearman Rank correlation is calculated between the number of layers and the average length of pattern retained over all configurations. At a first glance, there is no consistent behaviour in any RNN when increasing the number of layers. One of the more peculiar aspects of this, is that the bidirectional RNNs do not have the same behaviour as their single directional counterparts. Adding a bidirectional component to the RNN fundamentally changes the utilisation of parameters. For each increase in layer a different RNN either benefits or degrades. However, if the averages are taken into account, it is clear that those RNNs which were applied to the regression formulation, have a more negative correlation to sequence retention with an increase in the number of layers. The RNNs applied to the classification experiment, have more positive correlations to increasing the number of layers and retention ratios. To make definitive conclusions on the effects of increasing the number of layers, the per parameter sequence capacity ratio at each layer for each RNN will be evaluated next.

RNN		Number	of	layers			
		1	2	3	4	5	Average
Bidirectional Elman	Classification	-0.0664	0.0249	0.0000	0.0080	0.0480	0.00290
	Regression	-0.0823	-0.2806	-0.0468	-0.0163	-0.0382	-0.09284
Bidirectional GRU	Classification	0.0006	0.4537	0.0580	0.0809	-0.2566	0.06732
	Regression	-0.2103	-0.2180	0.0240	-0.0228	-0.0496	-0.09534
Bidirectional Jordan	Classification	0.0000	0.0015	0.0000	0.0000	0.0000	0.00030
	Regression	-0.0991	-0.0853	0.0220	0.0277	0.0059	-0.02576
Bidirectional LSTM	Classification	-0.0505	0.3804	0.0720	0.0046	-0.0923	0.06284
	Regression	-0.1811	0.1629	0.0063	0.0791	-0.0648	0.00048
Elman	Classification	-0.0345	0.1763	-0.1563	-0.0557	-0.0998	-0.03400
	Regression	-0.0780	-0.1096	0.0080	0.0305	-0.0128	-0.03238
GRU	Classification	0.0000	0.2788	-0.0911	0.1836	0.0338	0.08102
	Regression	-0.1990	0.1929	-0.0511	-0.0091	0.0153	-0.01020
Jordan	Classification	0.0531	0.2956	0.0583	0.0507	-0.1613	0.05928
	Regression	-0.0197	0.3212	0.0356	-0.0513	0.0762	0.07240
LSTM	Classification	0.0000	0.0000	0.0766	-0.0023	-0.1146	-0.00806
	Regression	-0.2020	0.2536	0.1041	-0.1100	-0.0336	0.00242

TABLE 7.6: Spearman correlation between number of patterns retained and the number of network parameters for increasing NN depth for each RNN

The findings of the effect increasing the number of hidden layers have on the sequence capacity per parameter, is illustrated in figures 7.45, 7.46, 7.47, 7.48, 7.49, 7.50, 7.51 and 7.52 will be referred to, followed by an inspection of table 7.7. The figures indicate that the per parameter capacity drastically drops from using one hidden layer to two. Thereafter the degradation plateaus as the number of hidden layers are increased. The degradation is more severe in the regression case than for classification. The degradation can be further seen in table 7.7 where, in the worst case, the sequence capacity per parameter in the regression formulation for the GRU decreased from 0.00094 to 0.00018, when transitioning from one hidden layer to two. As illustrated by table 7.7 it is clear to see as the number of layers increase, so does the per parameter sequence length capacity decrease. When increasing the number the retention ratio of each parameter goes down as the number of parameters increases in the network, however a loss of precision happens between layers and is only exaggerated as the depth of the network increases.

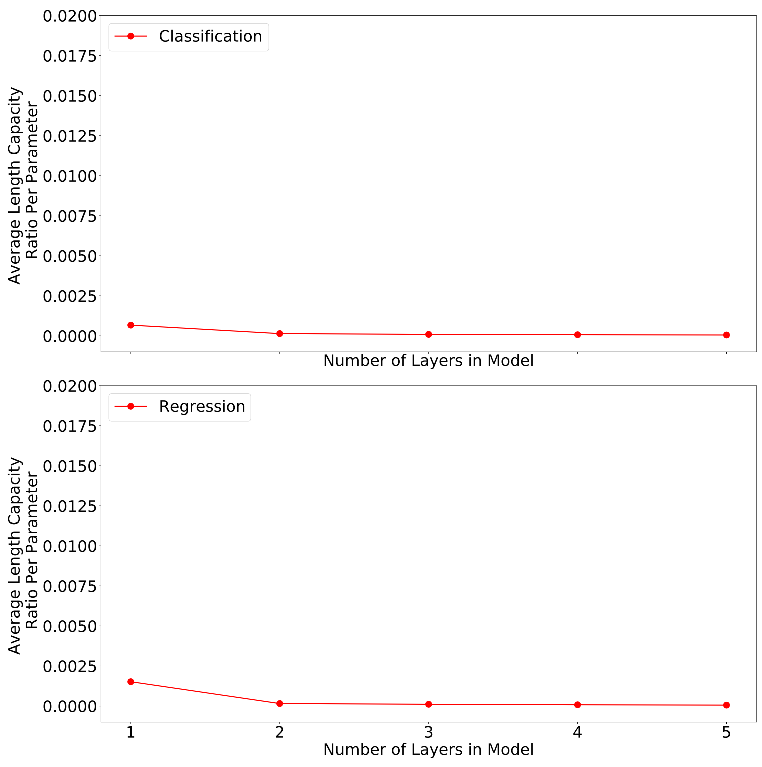

FIGURE 7.45: **Effect of increasing the number of parameters on the per parameter capacity of the network when increasing the number of layers for the LSTM RNN**

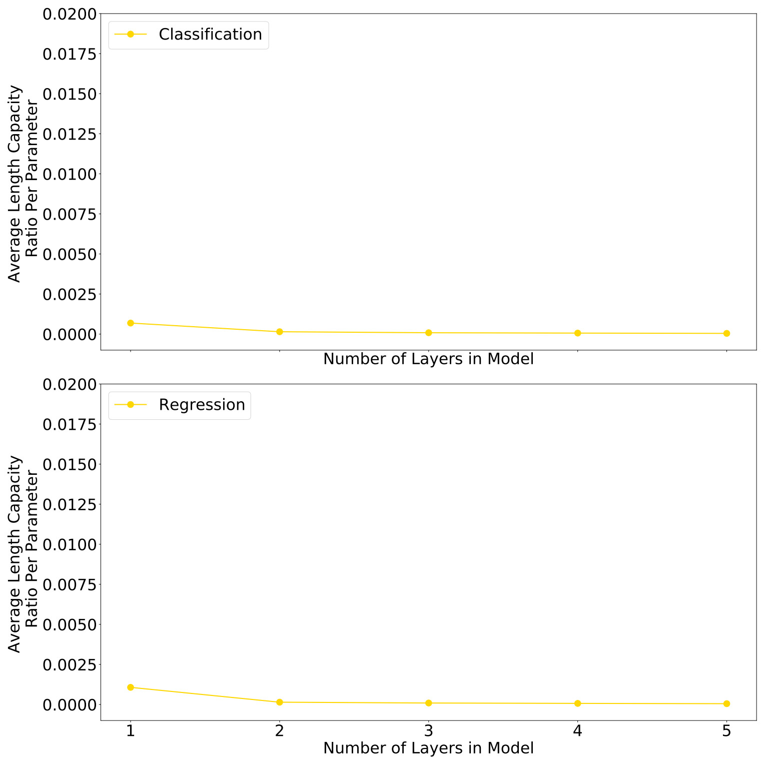

Figure 7.46: **Effect of increasing the number of parameters on the per parameter capacity of the network when increasing the number of layers for the Jordan RNN**

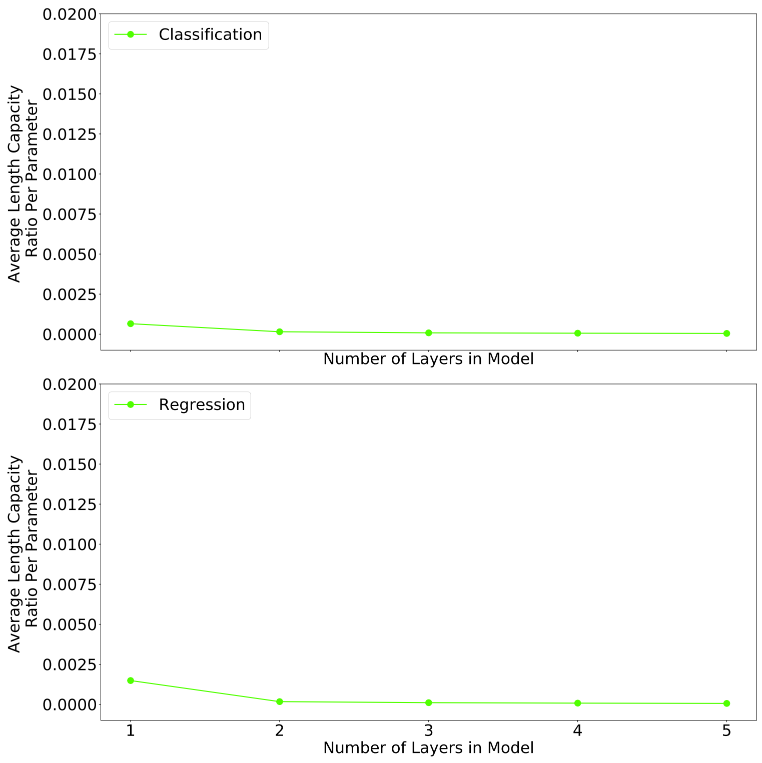

Figure 7.47: **Effect of increasing the number of parameters on the per parameter capacity of the network when increasing the number of layers for the Bidirectional Jordan RNN**

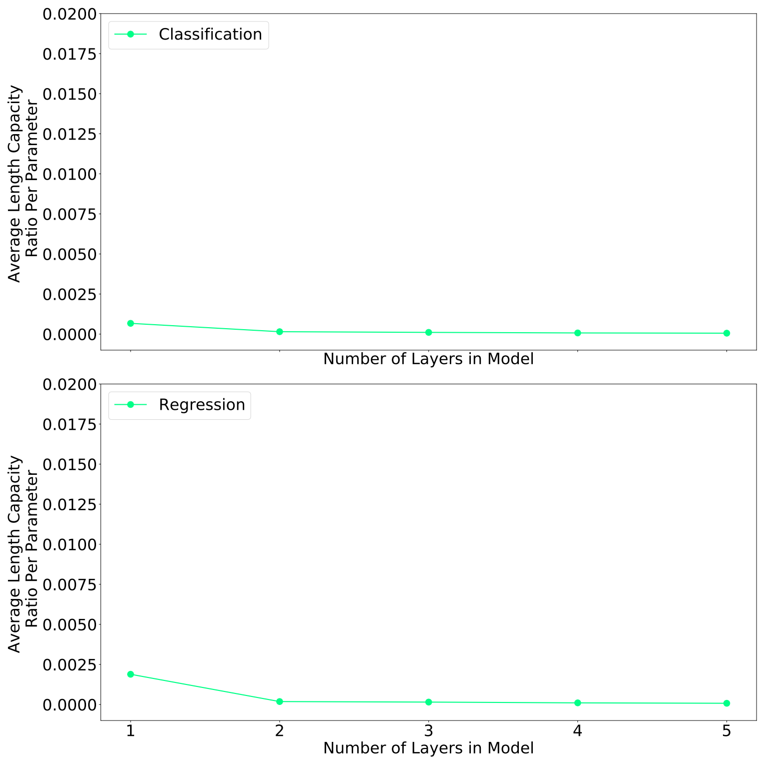

FIGURE 7.48: **Effect of increasing the number of parameters on the per parameter capacity of the network when increasing the number of layers for the GRU RNN**

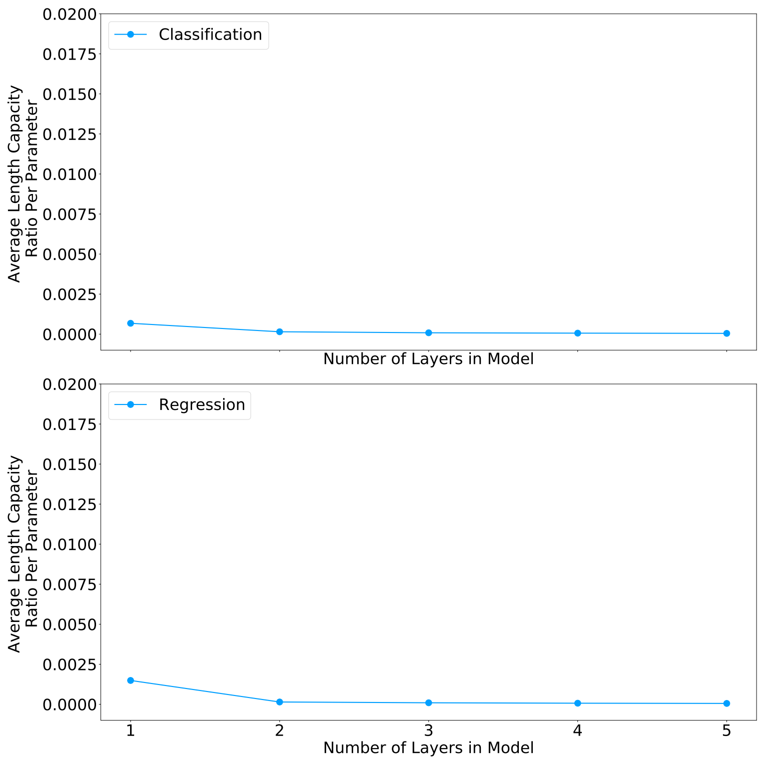

FIGURE 7.49: **Effect of increasing the number of parameters on the per parameter capacity of the network when increasing the number of layers for the Elman RNN**

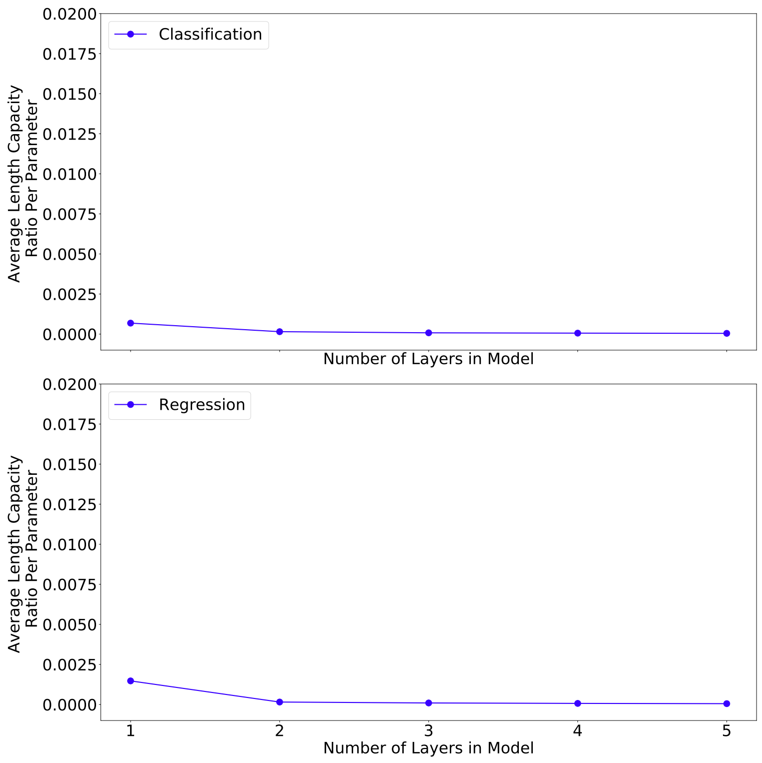

FIGURE 7.50: **Effect of increasing the number of parameters on the per parameter capacity of the network when increasing the number of layers for the Bidirectional Elman RNN**

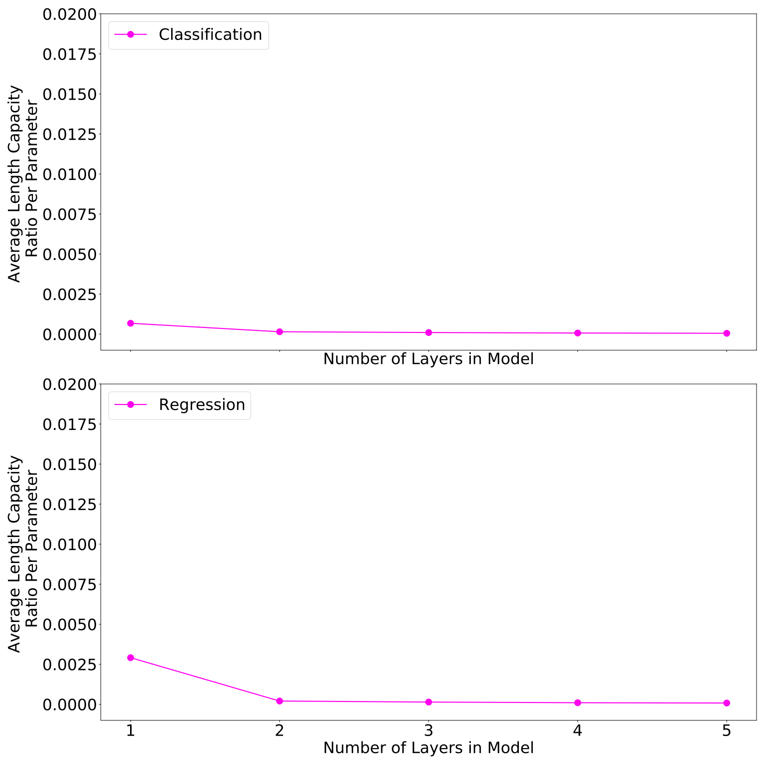

FIGURE 7.51: **Effect of increasing the number of parameters on the per parameter capacity of the network when increasing the number of layers for the Bidirectional LSTM RNN**

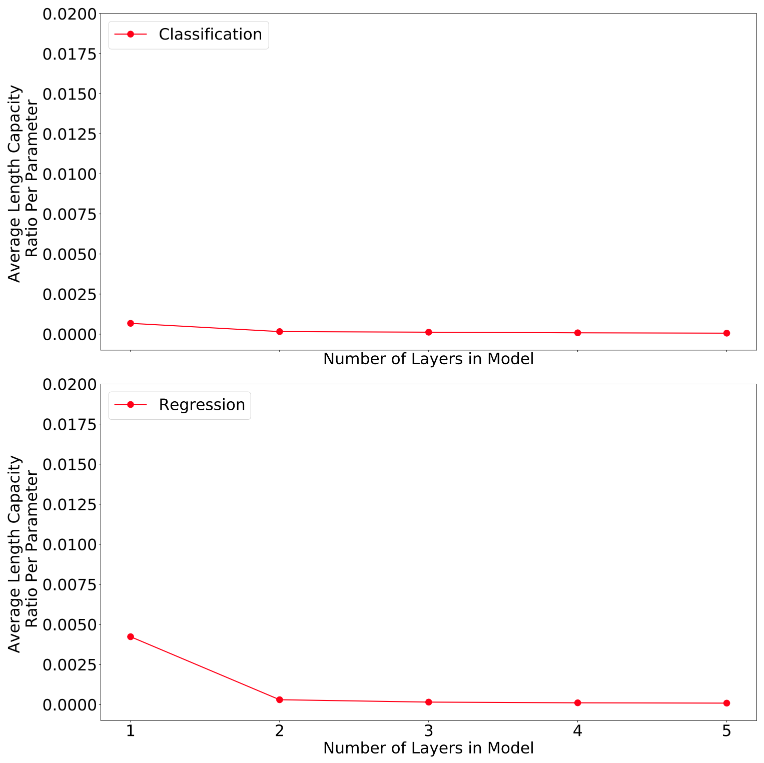

FIGURE 7.52: **Effect of increasing the number of parameters on the per parameter capacity of the network when increasing the number of layers for the Bidirectional GRU RNN**

RNNs		Number of layers 1	2	3	4	5	Average
Bidirectional Elman	Classification	0.00040	0.00015	0.00008	0.00006	0.00005	0.00015
	Regression	0.00087	0.00015	0.00010	0.00007	0.00005	0.00025
Bidirectional GRU	Classification	0.00039	0.00016	0.00012	0.00008	0.00006	0.00016
	Regression	0.00215	0.00028	0.00015	0.00010	0.00009	0.00055
Bidirectional Jordan	Classification	0.00038	0.00014	0.00008	0.00006	0.00004	0.00014
	Regression	0.00079	0.00016	0.00010	0.00007	0.00005	0.00023
Bidirectional LSTM	Classification	0.00040	0.00015	0.00010	0.00007	0.00005	0.00015
	Regression	0.00144	0.00020	0.00014	0.00010	0.00008	0.00039
Elman	Classification	0.00040	0.00014	0.00008	0.00006	0.00005	0.00015
	Regression	0.00082	0.00014	0.00009	0.00007	0.00005	0.00023
GRU	Classification	0.00040	0.00015	0.00011	0.00007	0.00006	0.00016
	Regression	0.00094	0.00018	0.00015	0.00010	0.00008	0.00029
Jordan	Classification	0.00041	0.00014	0.00008	0.00006	0.00005	0.00015
	Regression	0.00068	0.00014	0.00009	0.00007	0.00005	0.00021
LSTM	Classification	0.00039	0.00014	0.00009	0.00007	0.00006	0.00015
	Regression	0.00069	0.00015	0.00011	0.00008	0.00006	0.00022

TABLE 7.7: Per parameter capacity for each RNN type at various layer count

7.1.4 Relationship between Sequence Length Retention and Different Training Algorithms

The last aspect of Experiment 1 under consideration is the relationship between the length of pattern retained per parameter and the optimisation algorithm used to train each of the RNN types.

To determine this relationship, the same format is followed as the previous experiments in this section. The average sequence length retained is compared to the number of parameters used in the RNNs using a specific training algorithm, as illustrated in figures 7.53, 7.54, 7.55 and 7.56. Here it is clear that the RPROP and Adam training algorithms enable RNNs to retain the longest sequence length per parameter. As found in sections 7.1.1 to 7.1.3 the regression formulation resulted in better sequence length retention.

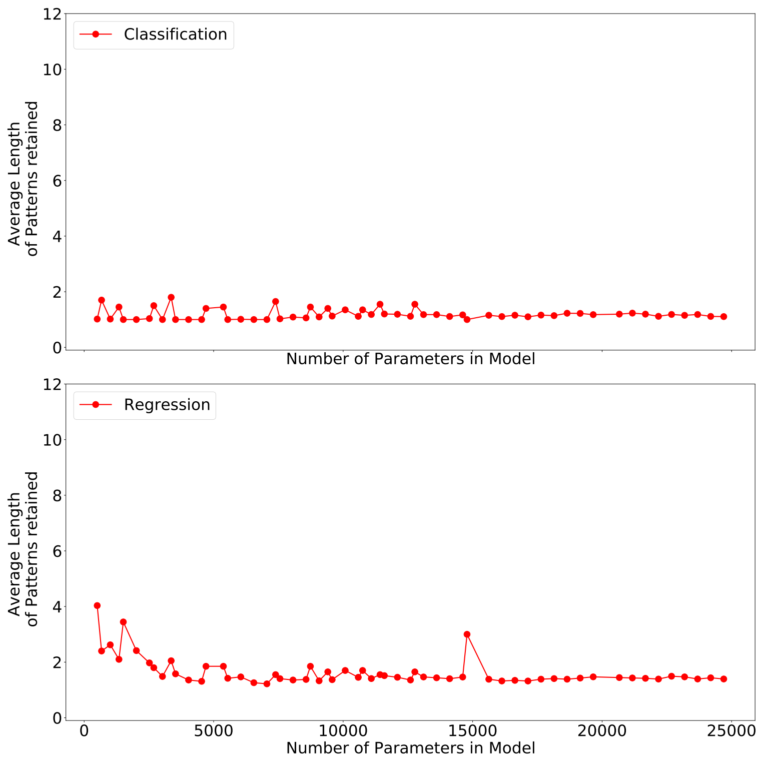

FIGURE 7.53: **Effect of using Adam training algorithm on the ability to retain patterns**

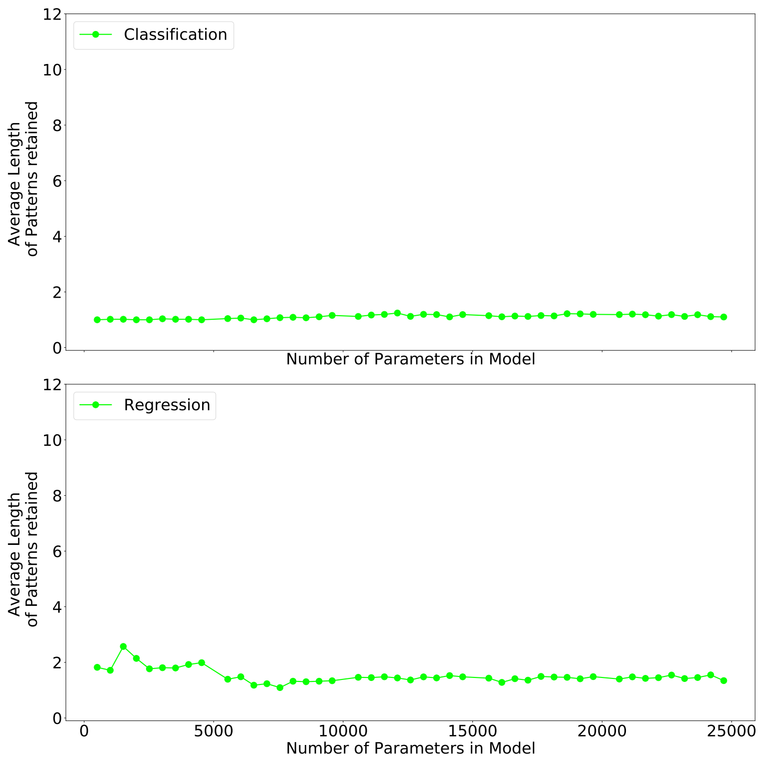

FIGURE 7.54: **Effect of using Nesterov training algorithm on the ability to retain patterns**

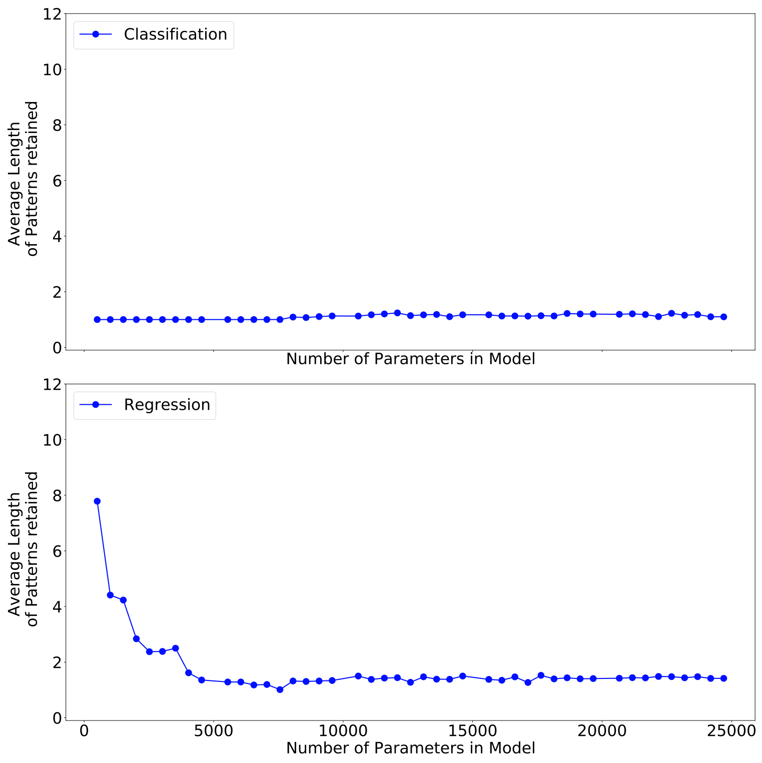

FIGURE 7.55: **Effect of using RPROP training algorithm on the ability to retain patterns**

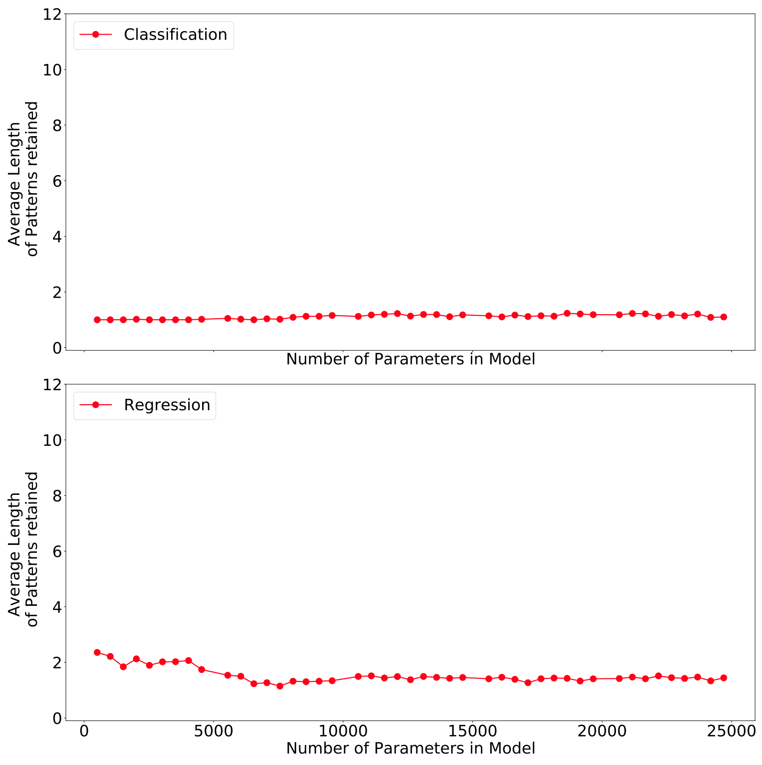

FIGURE 7.56: **Effect of using SGD training algorithm on the ability to retain patterns**

Continuing the analysis, the correlation of the length of sequence retained given each training algorithm for each RNN is presented in table 7.8. From table 7.8, the largest correlation between number of parameters and the longest sequence retained can be seen to be that of the bidirectional GRU using the Adam training algorithm, followed by the bidirectional GRU using RPROP. However, each RNN type seems to favour different training algorithms depending on the experiment type. For example the bidirectional LSTM favoured standard gradient descent (SDG) achieving the highest Spearman rank correlation coefficient of 0.3198 using the Nesterov optimisation algorithm in the regression formulation and 0.3015 using the SDG optimisation algorithm in the classification formulation.

		Training	algorithms		
RNN		Adam	Nesterov	RPROP	SGD
Bidirectional Elman	Classification	0.0792	0.0511	0.1594	0.0765
	Regression	-0.0487	-0.2697	0.0945	-0.2741
Bidirectional GRU	Classification	0.4476	0.4142	0.4192	0.4110
	Regression	-0.4367	0.3331	-0.4119	0.3384
Bidirectional Jordan	Classification	-0.0013	-0.0480	-0.0098	-0.0350
	Regression	-0.0877	-0.2265	-0.0546	-0.2646
Bidirectional LSTM	Classification	0.2706	0.2917	0.2833	0.3015
	Regression	-0.2770	0.3198	-0.3086	0.2308
Elman	Classification	0.1714	0.0365	0.1532	0.1271
	Regression	0.0061	-0.2817	0.2166	-0.2602
GRU	Classification	0.4087	0.3753	0.3890	0.3796
	Regression	-0.1459	0.3332	-0.2276	0.3302
Jordan	Classification	-0.0489	0.0361	0.1558	0.0807
	Regression	0.0123	-0.2370	0.2249	-0.2704
LSTM	Classification	0.3714	0.3574	0.3517	0.3549
	Regression	-0.1455	0.0413	-0.1483	0.0110

Table 7.8: Spearman correlation between number of patterns retained and the number of network parameters using different training algorithms for each RNN

The effect of different training algorithms on the various RNN types is illustrated in figures 7.57, 7.58, 7.59 and 7.60.

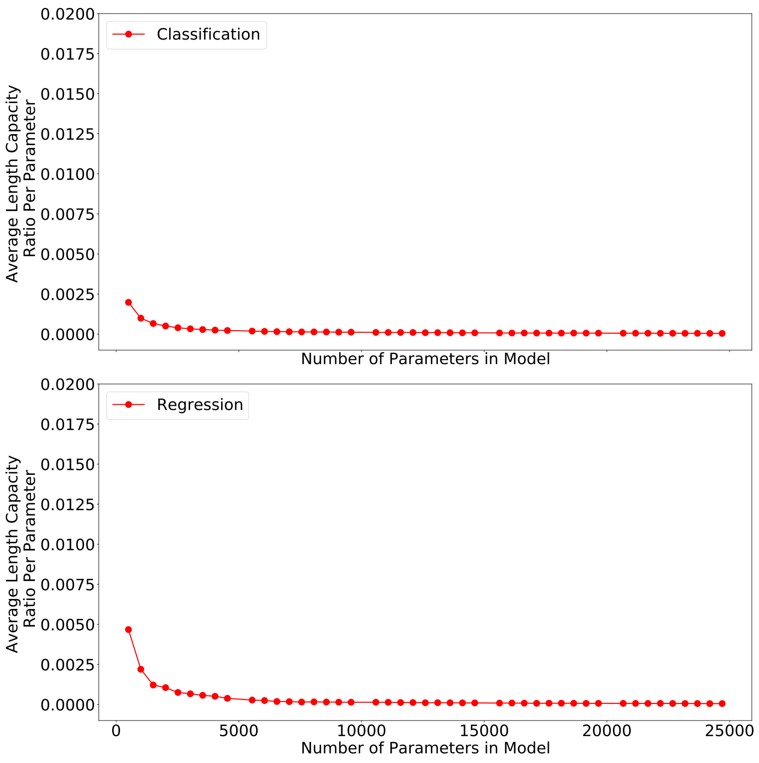

FIGURE 7.57: **Effect of increasing the number of parameters on the per parameter capacity of the network using the SGD training algorithm**

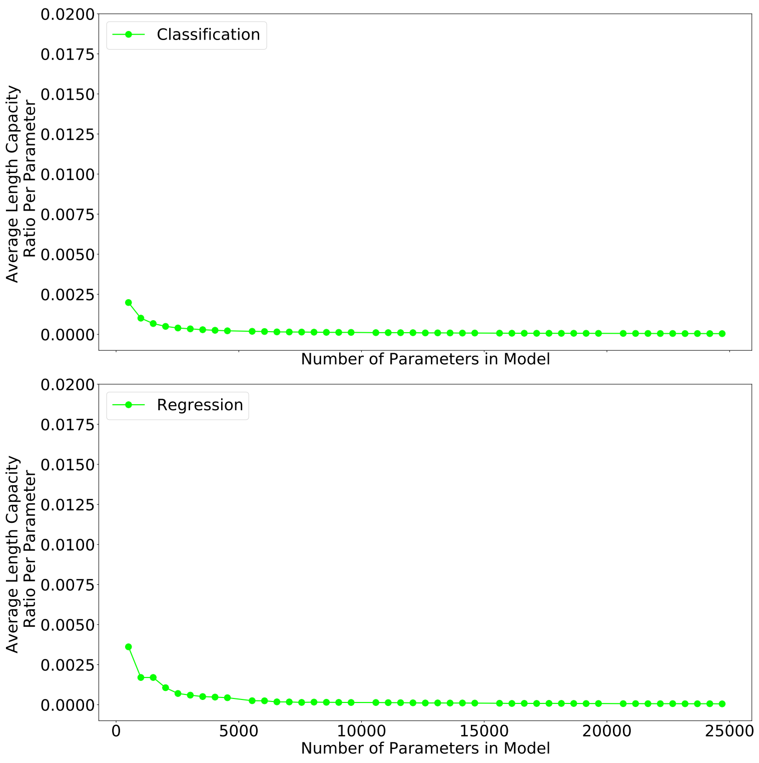

FIGURE 7.58: **Effect of increasing the number of parameters on the per parameter capacity of the network using the Nesterov training algorithm**

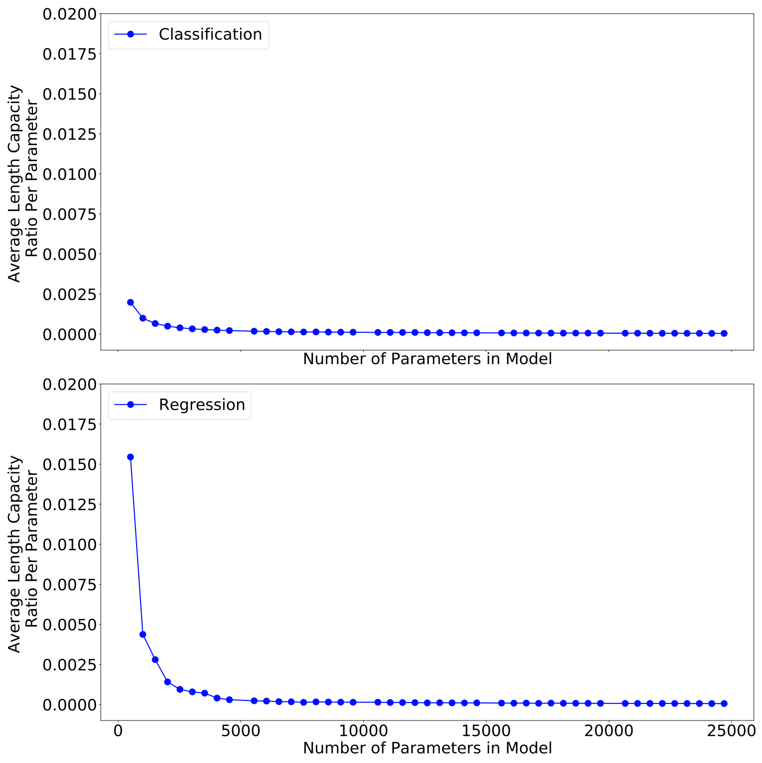

FIGURE 7.59: **Effect of increasing the number of parameters on the per parameter capacity of the network using the RPROP training algorithm**

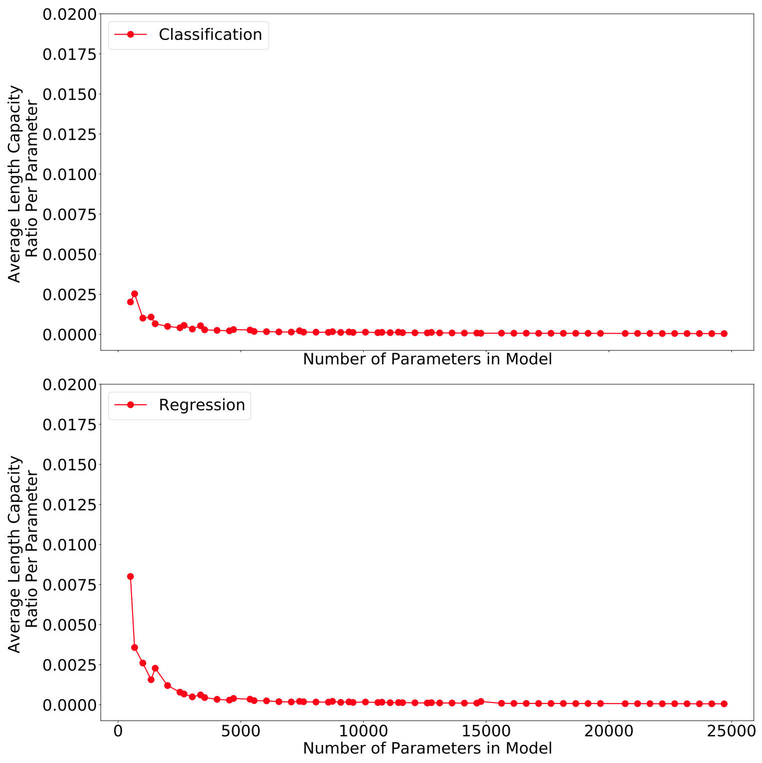

FIGURE 7.60: **Effect of increasing the number of parameters on the per parameter capacity of the network using the Adam training algorithm**

From figures 7.57, 7.58, 7.59 and 7.60 previous claims are validated; both RPROP and the Adam training algorithms result in the largest per parameter sequence capacity. To determine which configuration had the best per parameter sequence capacity ratio, table 7.9 will be further analysed.

		Training algorithms			
RNN		Adam	Nesterov	RPROP	SGD
Bidirectional Elman	Classification	0.00013	0.00011	0.00010	0.00011
	Regression	0.00013	0.00017	0.00011	0.00016
Bidirectional GRU	Classification	0.00015	0.00013	0.00013	0.00013
	Regression	0.00044	0.00016	0.00043	0.00016
Bidirectional Jordan	Classification	0.00012	0.00011	0.00011	0.00011
	Regression	0.00014	0.00017	0.00013	0.00019
Bidirectional LSTM	Classification	0.00014	0.00012	0.00012	0.00012
	Regression	0.00027	0.00015	0.00035	0.00015
Elman	Classification	0.00013	0.00011	0.00011	0.00011
	Regression	0.00014	0.00017	0.00011	0.00019
GRU	Classification	0.00014	0.00012	0.00012	0.00012
	Regression	0.00025	0.00015	0.00028	0.00015
Jordan	Classification	0.00013	0.00011	0.00011	0.00011
	Regression	0.00013	0.00016	0.00011	0.00016
LSTM	Classification	0.00014	0.00012	0.00012	0.00012
	Regression	0.00018	0.00014	0.00017	0.00013

TABLE 7.9: Per parameter capacity between number of patterns retained and the number of network parameters using different training algorithms for each RNN

Table 7.9 presents the average per parameter capacity ratio achieved by each RNN type using specific training algorithms for both the classification and regression experiments. From table 7.9, the best per parameter sequence capacity ratio is achieved using the Adam training algorithm for the bidirectional GRU, followed by the bidirectional LSTM trained using RPROP for both the regression and classification variants of the experiment. When comparing the results between using the Adam and RPROP optimiser for the bidirectional GRU, there is only a slight increase when using the Adam training algorithm. The observed behaviour may be attributed to the stochastic nature of neural networks training, thus no best training algorithm can be definitively determined. However, considering the bidirectional LSTM the disparity between capacity ratio using the Adam and RPROP training algorithms is more apparent. The disparity in performance of the bidirectional LSTM trained using the Adam and RPROP training algorithms would indicate that the RPROP training algorithm is more suited to train RNNs if the goal is to optimise per sequence capacity ratio for each parameter used in a RNN, based on the current experimentation.

7.2 Experiment 2: Number of patterns retention

The analysis in this section follows the same format as the analysis done in section 7.1. Here the investigation looks at the total number of patterns retained by each network and the per parameter pattern capacity ratio. This experiment has two components, a regression and classification formulation of the problem statement as described in section 6.2.2.

The results are presented as follows: The average number of patterns retained by each RNN is presented in section 7.2.1. This is followed by investigating the use of different activation functions evaluated in section 7.2.2. Thereafter the relationship between the per parameter pattern capacity and the number of hidden layers is investigated in section 7.2.3. Closing this section off, the influence of different training algorithms on the number of pattern retained is inspected in section 7.2.4.

7.2.1 Average Per Parameter Capacity Retention for each RNN

The average number of patterns retained by each RNN is presented in figures 7.61, 7.62, 7.63, 7.64, 7.65, 7.66, 7.67 and 7.68. Similarly to the sequence retention experiment, for all the RNNs, the total number of patterns retained was better when using the regression formulation of the experiment as opposed to the classification formulation of the experiment. This could be a coincidence, however the results of this warrant further investigation. As a speculation, the performance dominance of the regression formulation over the classification may be the way in which the classification formulation is structured. If the classification formulation was structured differently, it may increase performance. However, a refactoring of the classification formulation can be investigated in future work.

From the figures 7.66 and 7.68, it is observed that the bidirectional LSTM and bidirectional GRU were able to more linearly increase their pattern retention as the number of parameters increased in the networks for the regression experiment. However, in the classification formulation the bidirectional LSTM and bidirectional GRU RNNs were more susceptible to local minima, as can be seen from figures 7.62 and 7.64 as the number of patterns retained varies between network configurations where in some cases, smaller network configurations out performs larger networks. For the Elman and Jordan RNNs, bidirectional layers did not increase the pattern retention ratios. Bidirectional layers helping the GRU and LSTM RNNs more than their less complex counterparts indicates that the GRU and LSTM RNNs have better capabilities in retaining an increasing number of patterns for this experiment and that the complex structures of the GRU and LSTM are justified. To investigate the relationship between the number of patterns retained and the increase in the number of parameters the Spearman rank correlation of this relationship will be scrutinised.

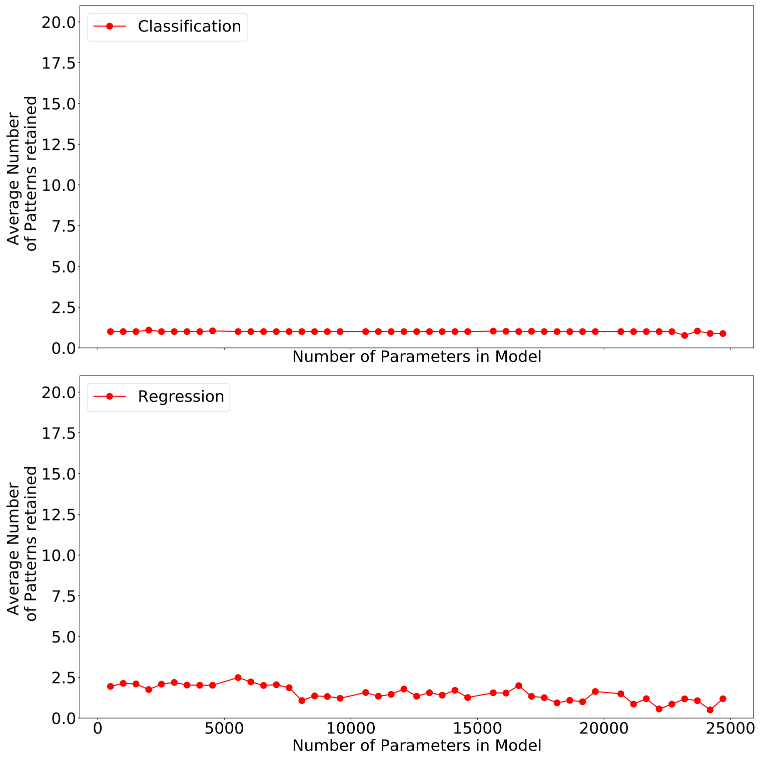

FIGURE 7.61: **Effect of the number of parameters on the ability to retain patterns for the bidirectional Elman RNN**

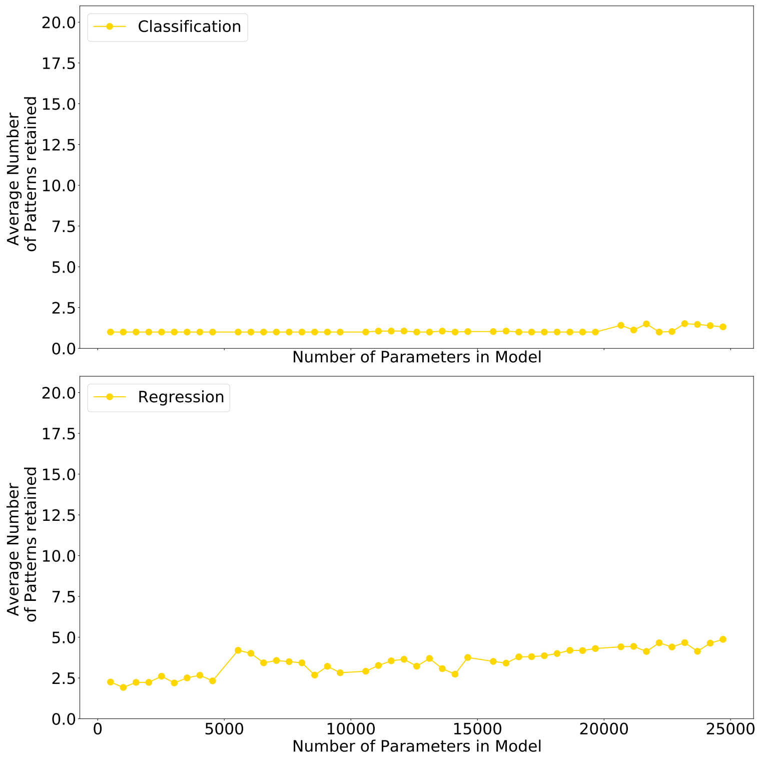

FIGURE 7.62: **Effect of the number of parameters on the ability to retain patterns for the bidirectional GRU RNN**

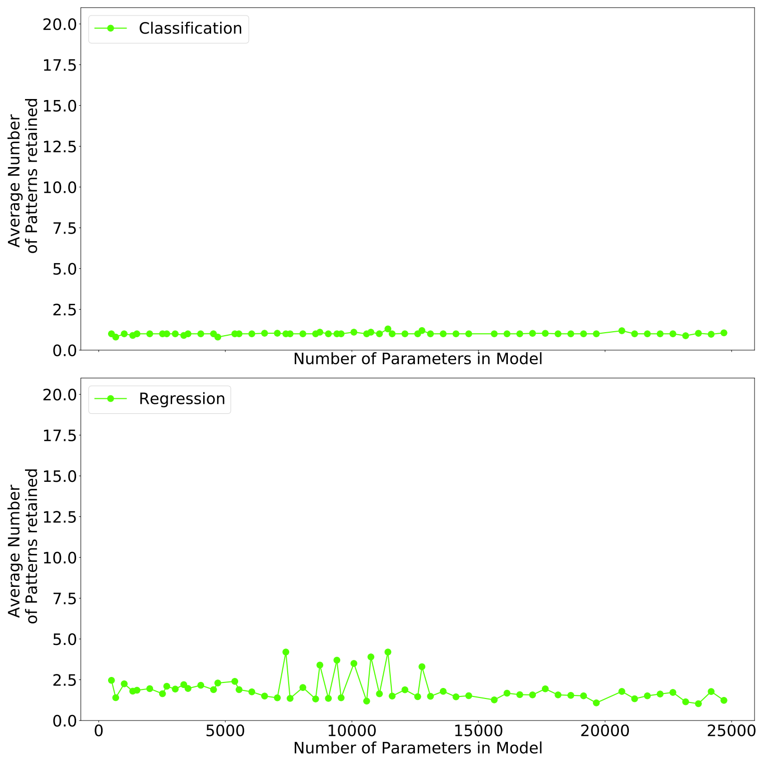

FIGURE 7.63: **Effect of the number of parameters on the ability to retain patterns for the bidirectional Jordan RNN**

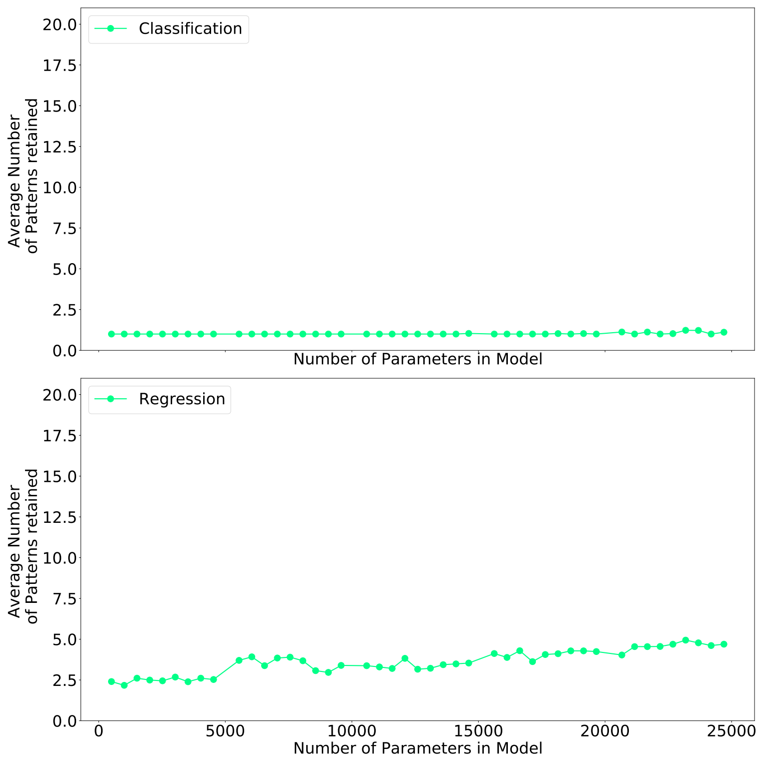

FIGURE 7.64: **Effect of the number of parameters on the ability to retain patterns for the bidirectional LSTM RNN**

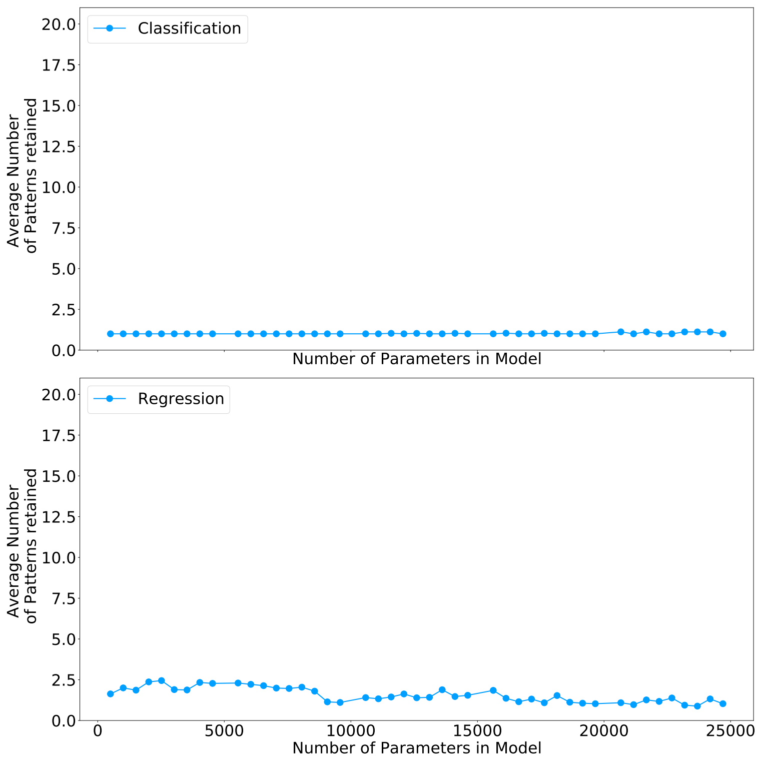

FIGURE 7.65: **Effect of the number of parameters on the ability to retain patterns for the Elman RNN**

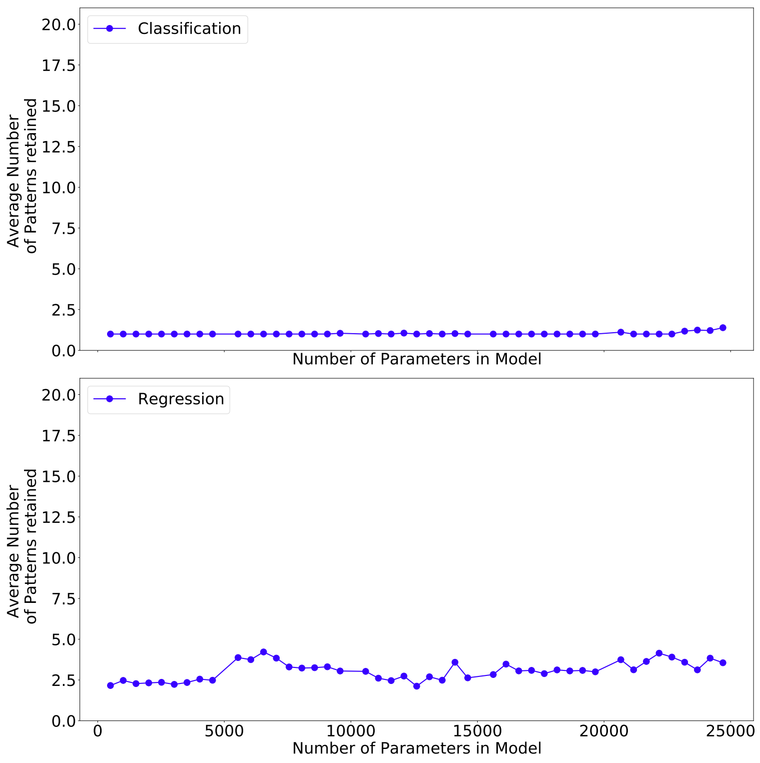

FIGURE 7.66: **Effect of the number of parameters on the ability to retain patterns for the GRU RNN**

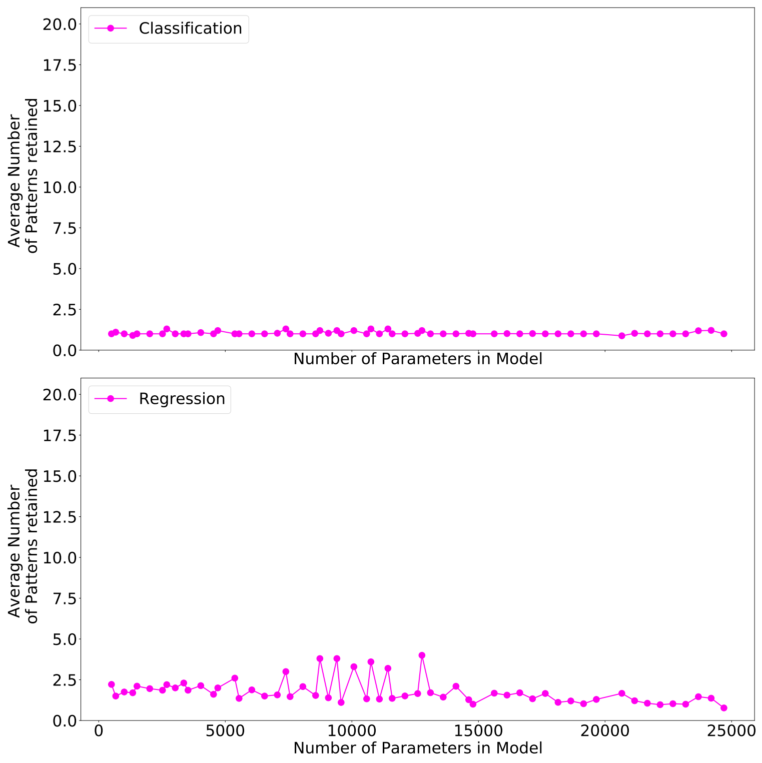

FIGURE 7.67: **Effect of the number of parameters on the ability to retain patterns for the Jordan RNN**

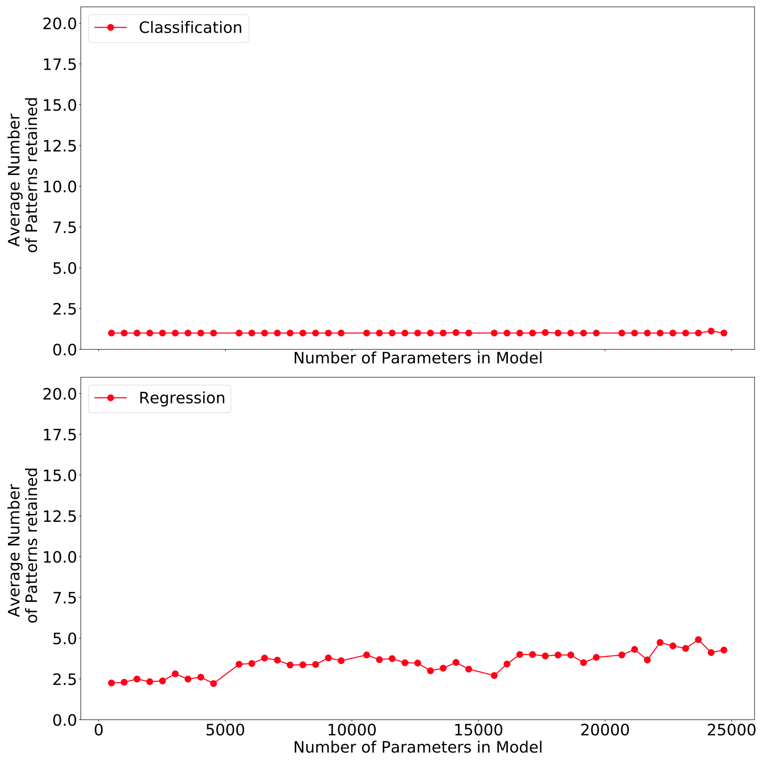

FIGURE 7.68: **Effect of the number of parameters on the ability to retain patterns for the LSTM RNN**

Table 7.10 explores the Spearman rank correlation between the number of patterns retained and the number of parameters in the various RNNs. The same behaviour is observed as in the figures 7.61, 7.62, 7.63, 7.64, 7.65, 7.66, 7.67 and 7.68 previously discussed, where the bidirectional GRU and bidirectional LSTM have the highest correlation between number of patterns retained and the number of parameters in the network in the regression experiment. For the classification experiment, the bidirectional GRU and GRU have the highest correlation. Upon inspecting the GRU and LSTM more closely, the LSTM has a higher correlation coefficient in the regression experiment whereas the GRU has a higher coefficient in the classification experiment. The Elman and Jordan RNNs coefficients indicate these RNNs are more suited for the classification experiment as compared to the regression experiment, which is investigated next.

RNN	Classification	Regression
Bidirectional Elman	-0.08784	-0.20633
Bidirectional GRU	0.30532	0.21190
Bidirectional Jordan	0.03103	-0.11337
Bidirectional LSTM	0.18325	0.20004
Elman	0.13788	-0.21105
GRU	0.18828	0.08863
Jordan	0.01996	-0.14631
LSTM	0.06968	0.16480

TABLE 7.10: Spearman correlation between number of patterns retained and the number of network parameters

To investigate the performance of the per parameter pattern capacity ratio, figures 7.69, 7.70, 7.71, 7.72, 7.73, 7.74, 7.75 and 7.76 will be evaluated followed by interrogating table 7.11. Upon inspecting these figures, it is clear that the per parameter pattern capacity ratio decreases rapidly in both the regression and classification experiments. The decrease in per parameter pattern capacity ratio was also observed in section 7.1.1. The plateauing of the degradation is just an effect of the ratio's in which these figures are presented. Thus, the degradation does not reach a set equilibrium where the increasing of the number of parameters does not further decrease the per parameter pattern capacity. Rather, the ratio approaches zero the more number of parameters are increased in the network. To conclusively determine which RNN had the best per parameter capacity ratio, refer to table 7.11.

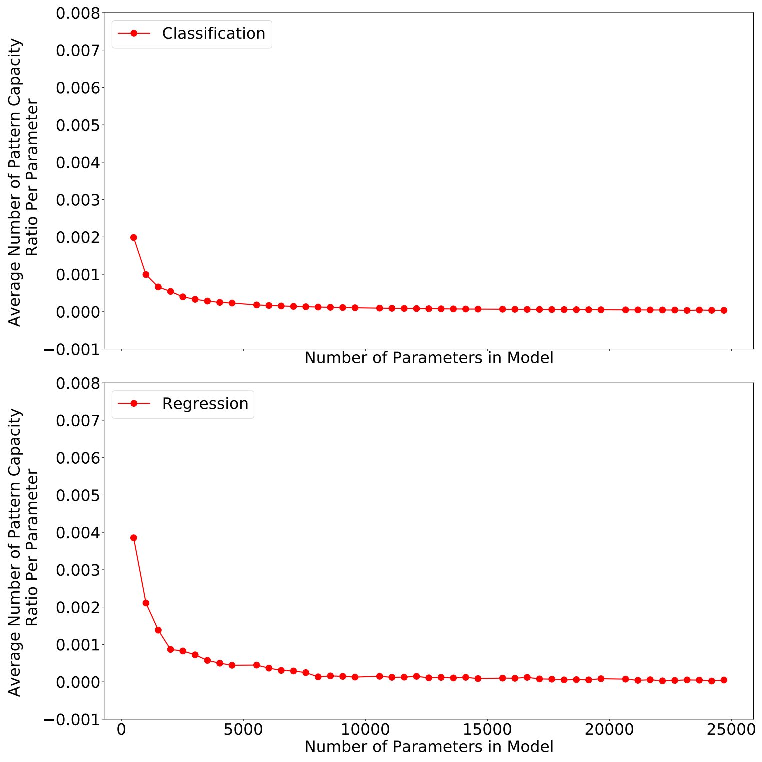

FIGURE 7.69: **Effect of increasing the number of parameters on the per parameter capacity of the Bidirectional Elman RNN**

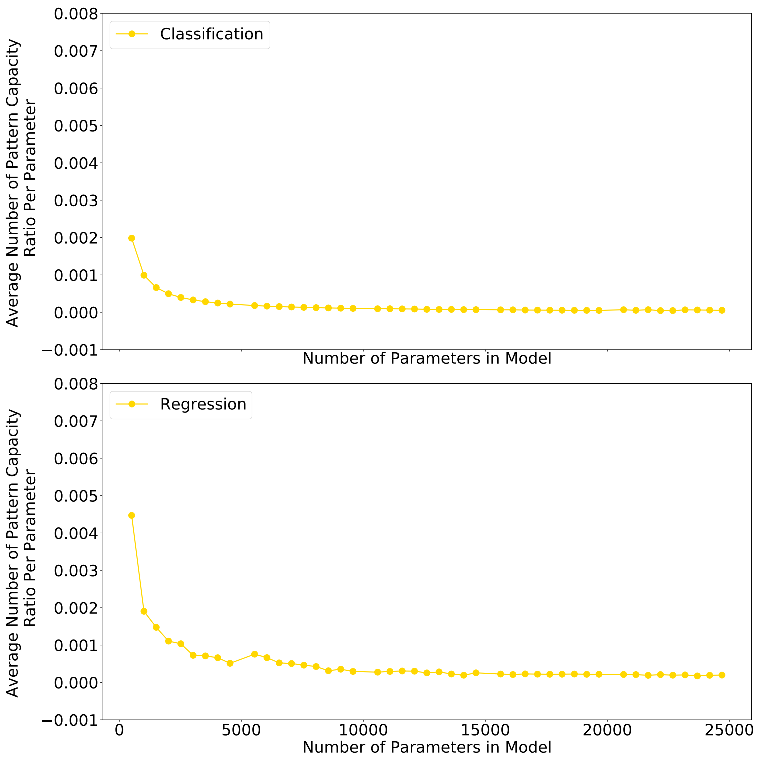

FIGURE 7.70: **Effect of increasing the number of parameters on the per parameter capacity of the Bidirectional GRU RNN**

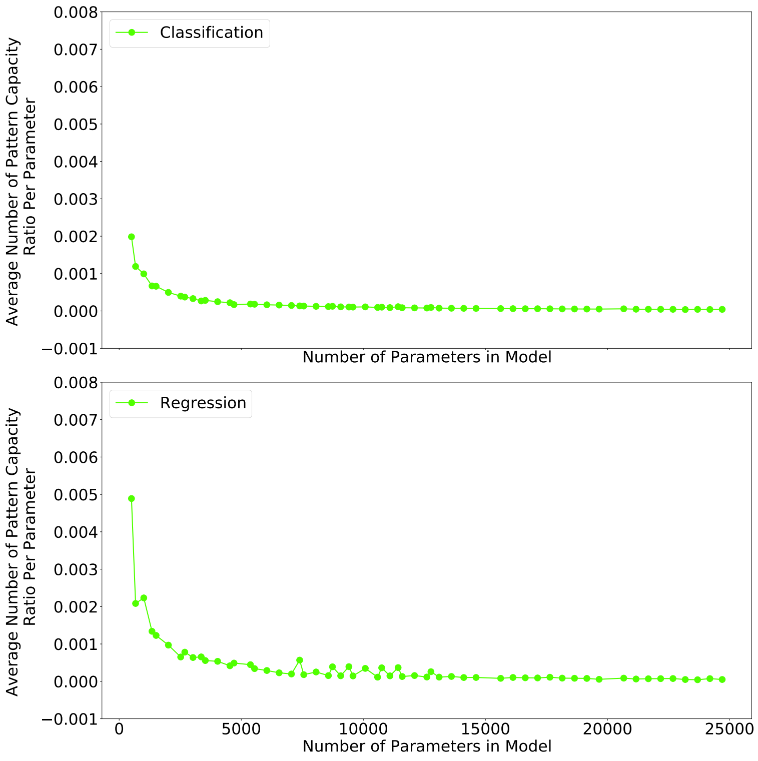

FIGURE 7.71: **Effect of increasing the number of parameters on the per parameter capacity of the Bidirectional Jordan RNN**

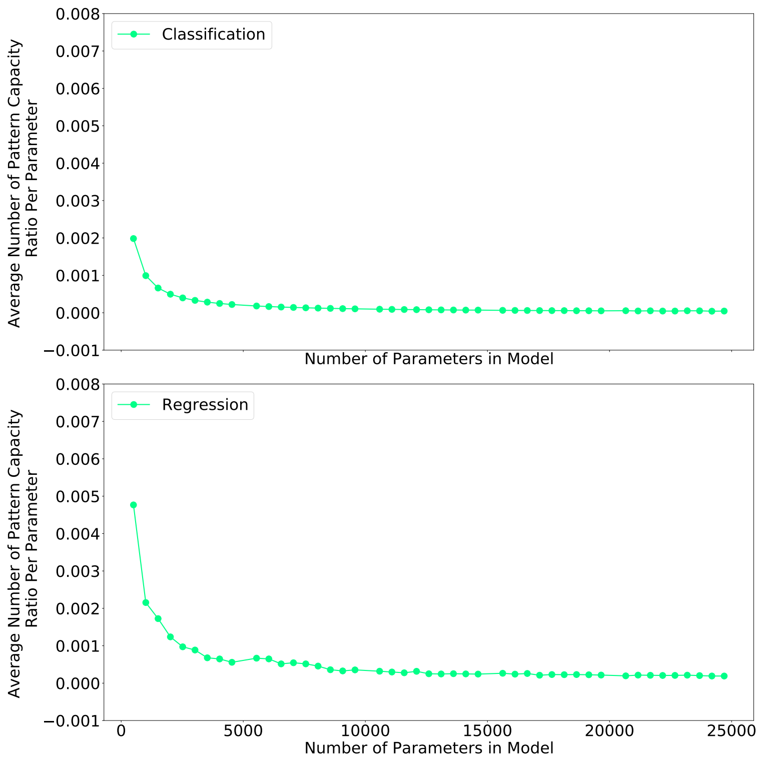

FIGURE 7.72: **Effect of increasing the number of parameters on the per parameter capacity of the Bidirectional LSTM RNN**

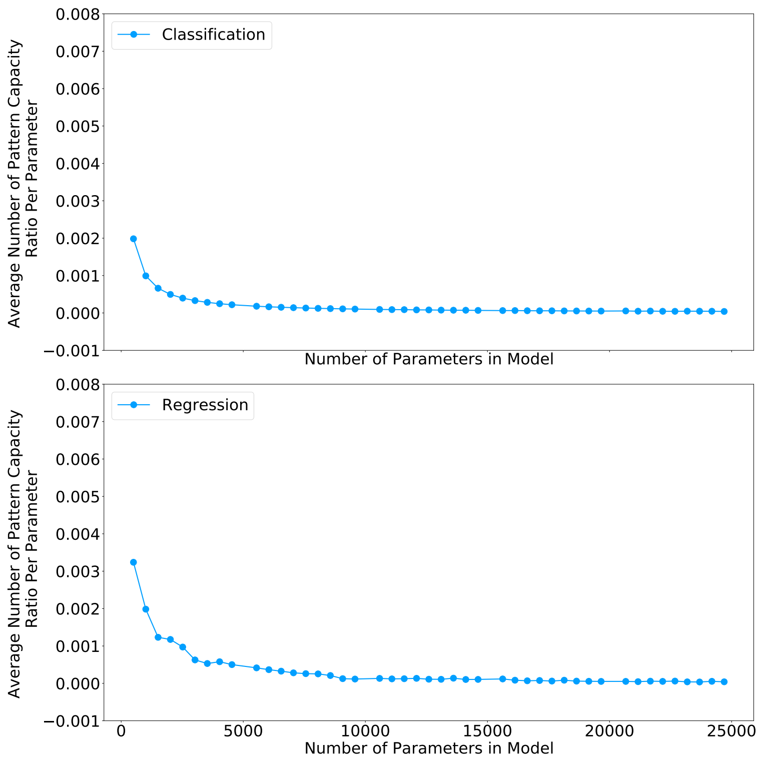

FIGURE 7.73: **Effect of increasing the number of parameters on the per parameter capacity of the Elman RNN**

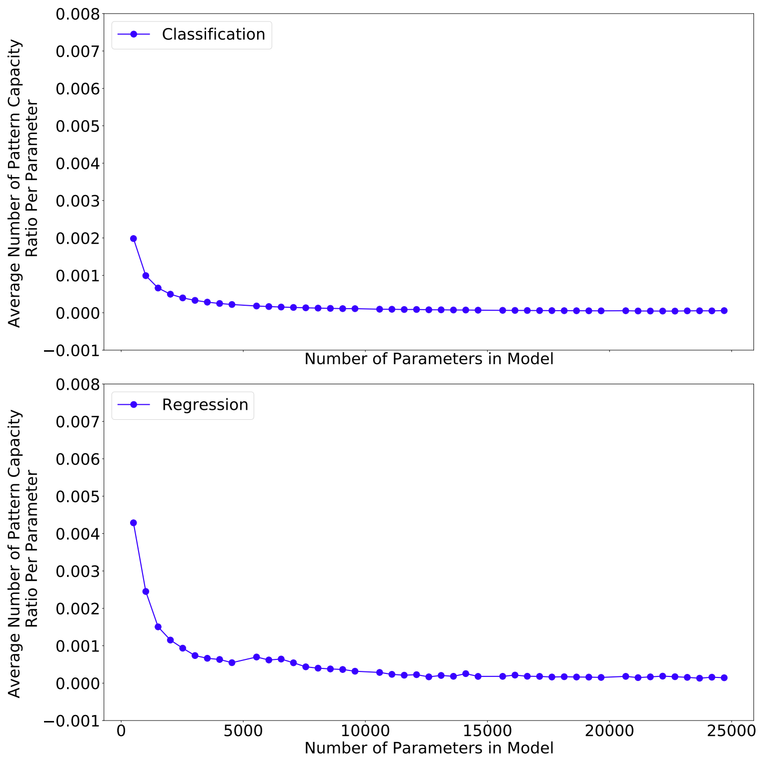

FIGURE 7.74: **Effect of increasing the number of parameters on the per parameter capacity of the GRU RNN**

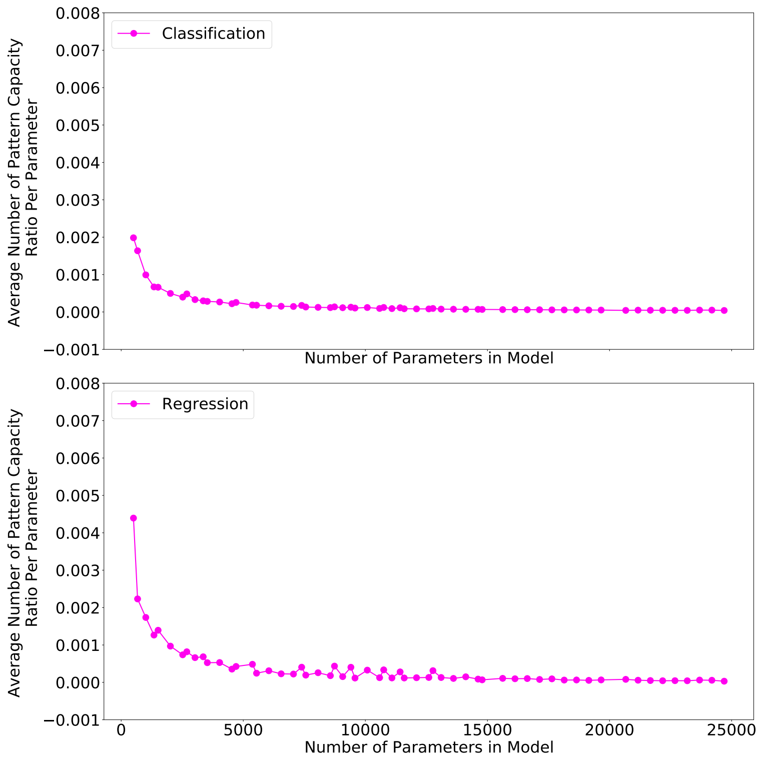

FIGURE 7.75: **Effect of increasing the number of parameters on the per parameter capacity of the Jordan RNN**

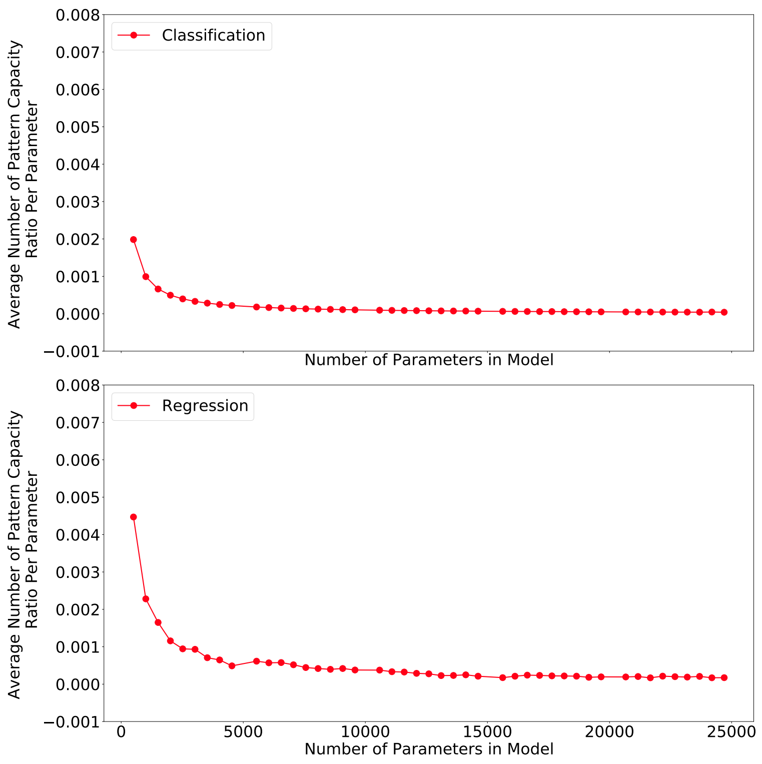

FIGURE 7.76: **Effect of increasing the number of parameters on the per parameter capacity of the LSTM RNN**

Table 7.11 represents the average per parameter capacity ratios for this experiment in both the regression and classification formulations. In the classification experiment the per parameter pattern capacity is similar for all RNNs, which is indicative that the problem formulation was difficult for all networks. However, if we investigate the regression case, it is clear that the bidirectional component assisted all but the Elman RNNs. The bidirectional layers thus, in general, increase the effectiveness of RNNs in using the number of parameters allotted to the network which may indicate that seeing the pattern in the opposite direction as well may have an advantageous effect on the updating of weights. In this experiment the bidirectional LSTM had the largest per parameter pattern capacity ratio of 0.0003331 patterns per parameter, followed by the LSTM RNN at 0.0003281. The LSTM RNN variants were more effective in this experiment than the others overall which may be attributed to the complexity of the internal state of the LSTM nodes which can keep track of more relevant information than other RNN nodes.

RNN	Classification	Regression
Bidirectional Elman	0.0000963	0.0001547
Bidirectional GRU	0.0001088	0.0003215
Bidirectional Jordan	0.0000973	0.0001552
Bidirectional LSTM	0.0001056	0.0003331
Elman	0.0001051	0.0001693
GRU	0.0001045	0.0002964
Jordan	0.0000966	0.0001474
LSTM	0.0001019	0.0003281

TABLE 7.11: Number of patterns retained per single network parameter

7.2.2 Average Per Parameter Capacity Retention for each Activation Function

As the overall performance of the RNNs were investigated in the previous section (section 7.2.1), this section analyses the impact various activation functions have on the effectiveness of pattern retention across RNNs.

The average number of patterns retained when using each activation function is presented in figures 7.77, 7.78, 7.79, 7.80, 7.81, 7.82, 7.83, 7.84, 7.85 and 7.86. Unsurprisingly, all under performed in the classification experiment. In the regression experiment, the results were more sporadic, fluctuating between retaining the patterns and not. As can be seen, the variance in patterns retained increases as the number of parameters increase for the elu, linear, selu, softmax and softplus activation functions. For the hard sigmoid and sigmoid the variance decreases as the number of parameters increase. However the number of patterns retained also decreased, indicating that RNNs using either hard sigmoid or sigmoid activation functions converge faster onto local optima than RNNs using other activation functions which may be due to the small range of these functions' search space which is constraint to $[0, 1)$ leading to faster vanishing gradients due to arithmetic errors in floating point rounding. The relu activation function had less variance overall outside of a few peaks, such as when 12 patterns were retained as can be observed in figure 7.80. The activation function which had the best increase in number of patterns retained is the softsign, which had a near linear increase in pattern retention. The pattern retention rate of the softsign may be due to this function having the range of $(-1, 1)$ which may help the neural network to detect irrelevant patterns more easily discard specific patterns previously learned.

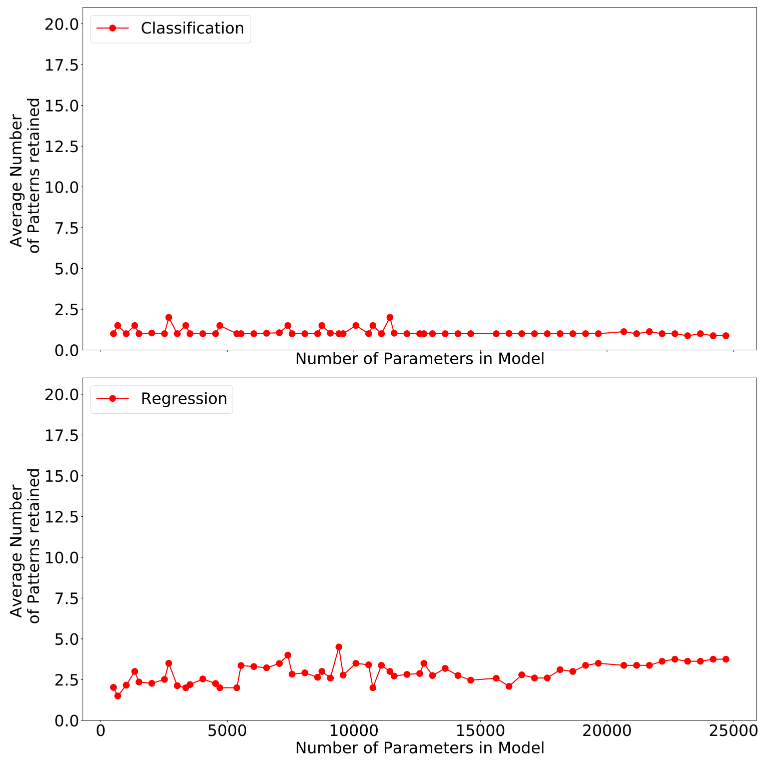

FIGURE 7.77: **Effect of the number of parameters on the ability to retain patterns over each activation function: elu**

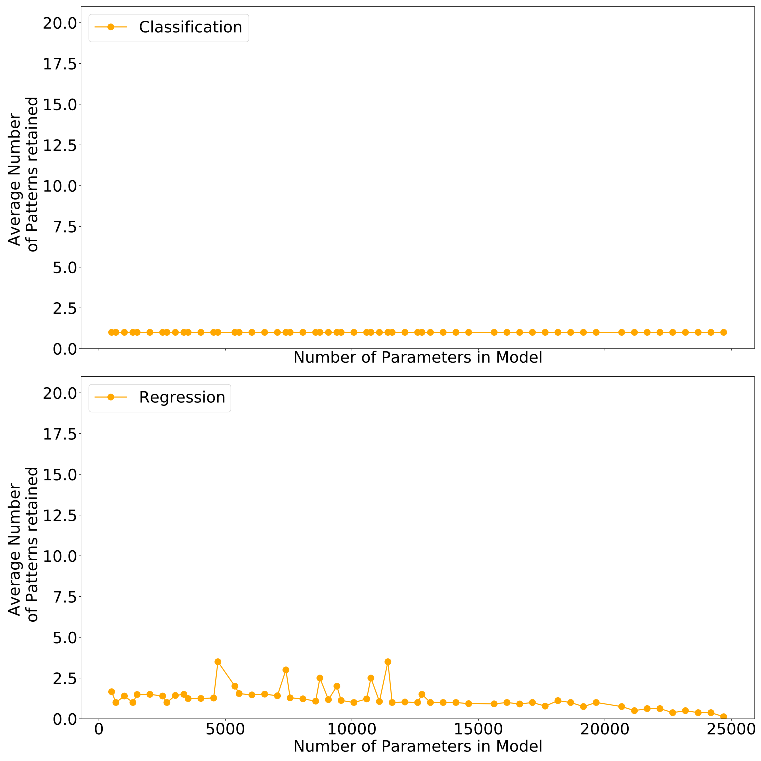

FIGURE 7.78: **Effect of the number of parameters on the ability to retain patterns over each activation function: hard sigmoid**

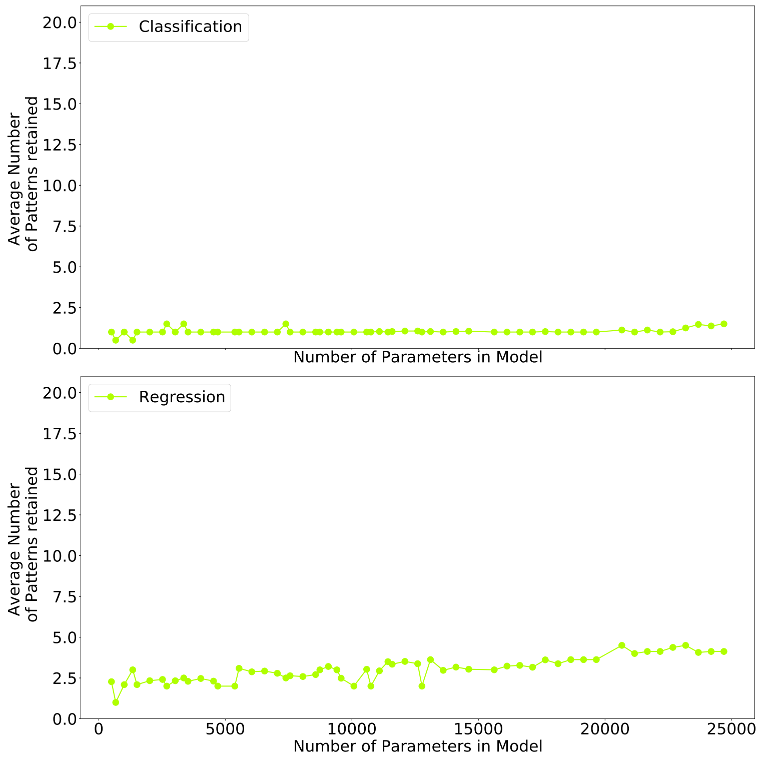

FIGURE 7.79: **Effect of the number of parameters on the ability to retain patterns over each activation function: linear**

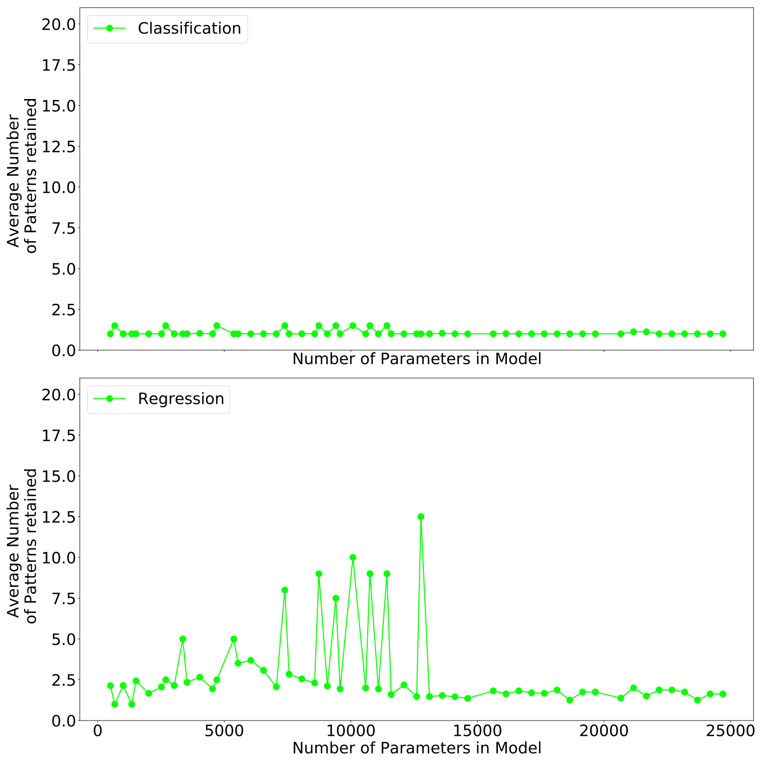

FIGURE 7.80: **Effect of the number of parameters on the ability to retain patterns over each activation function: relu**

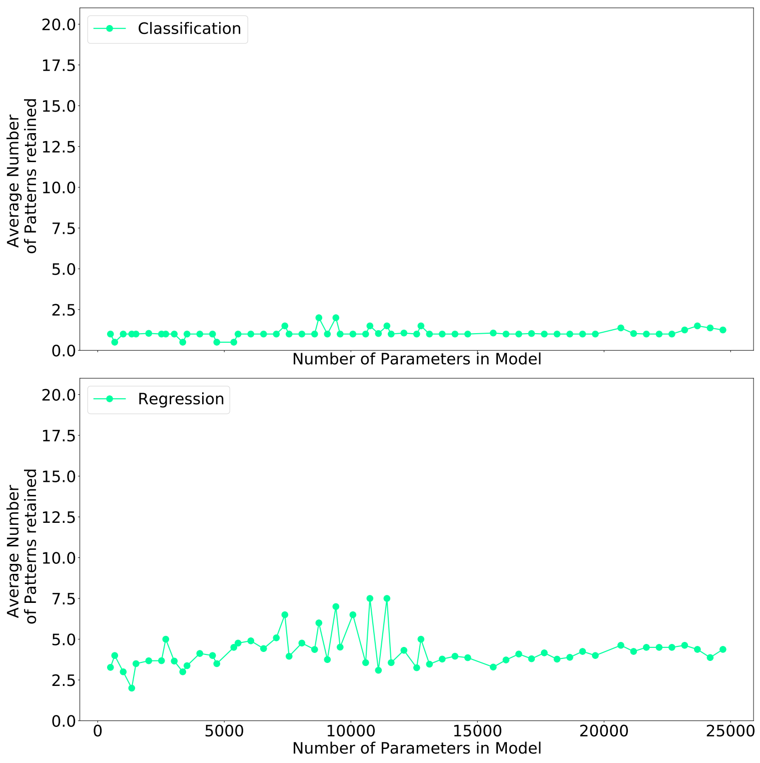

FIGURE 7.81: **Effect of the number of parameters on the ability to retain patterns over each activation function: selu**

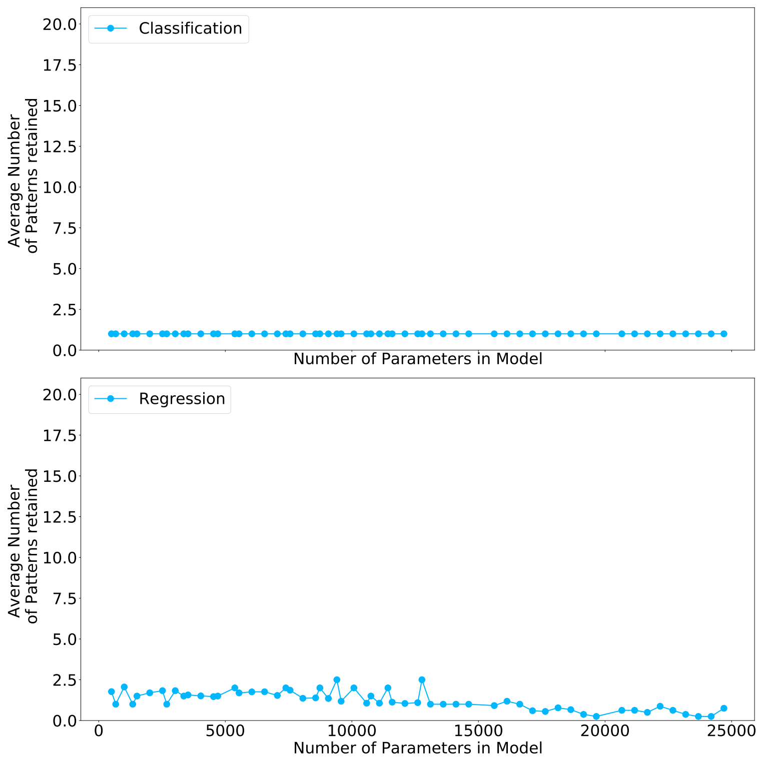

FIGURE 7.82: **Effect of the number of parameters on the ability to retain patterns over each activation function: sigmoid**

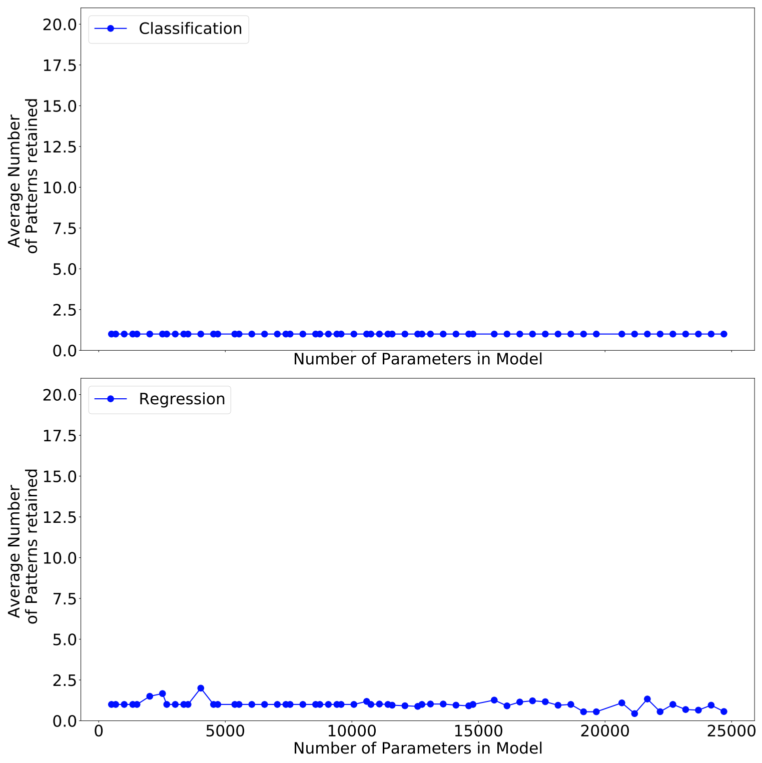

FIGURE 7.83: **Effect of the number of parameters on the ability to retain patterns over each activation function: softmax**

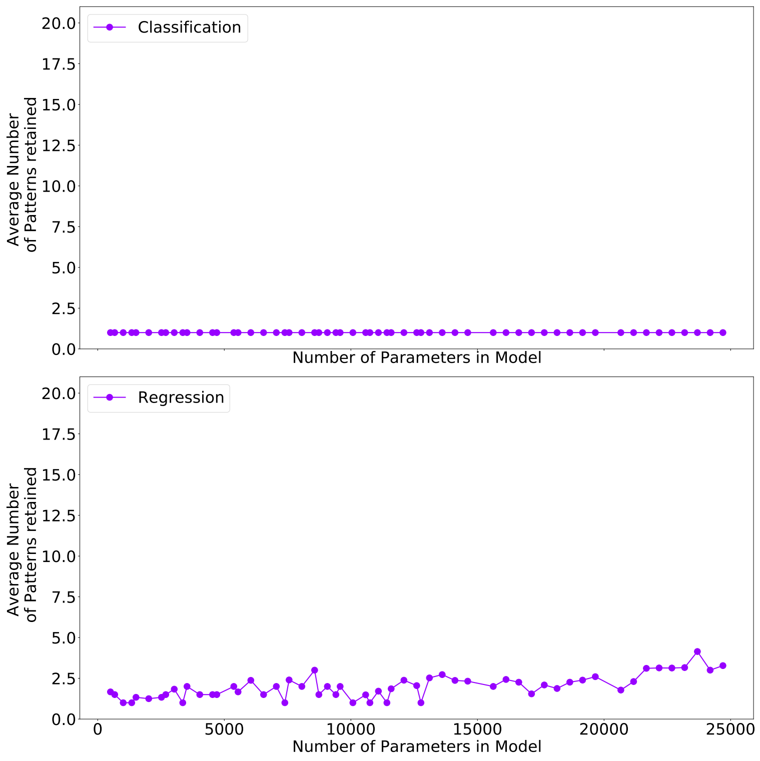

FIGURE 7.84: **Effect of the number of parameters on the ability to retain patterns over each activation function: softplus**

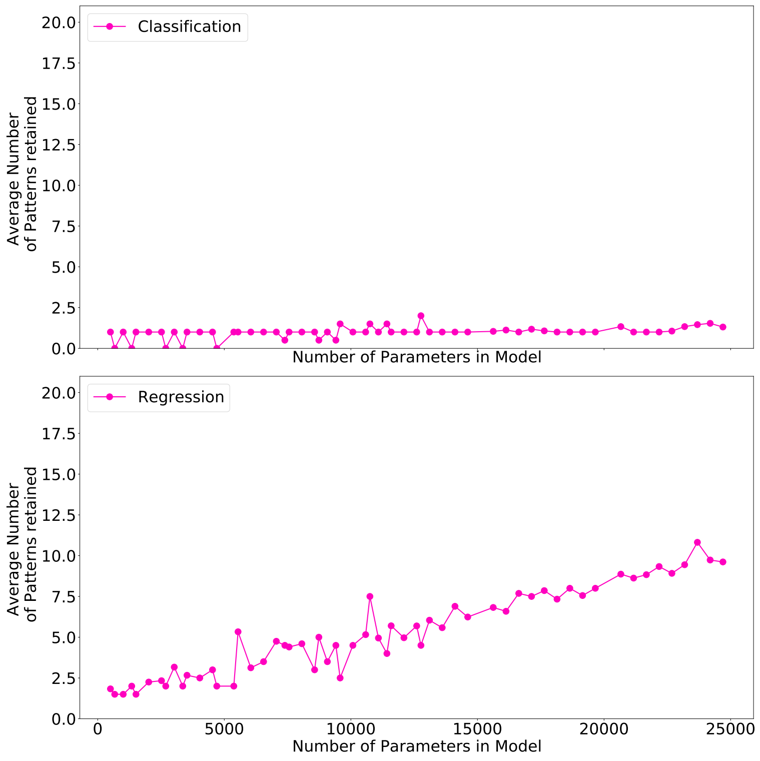

FIGURE 7.85: **Effect of the number of parameters on the ability to retain patterns over each activation function: softsign**

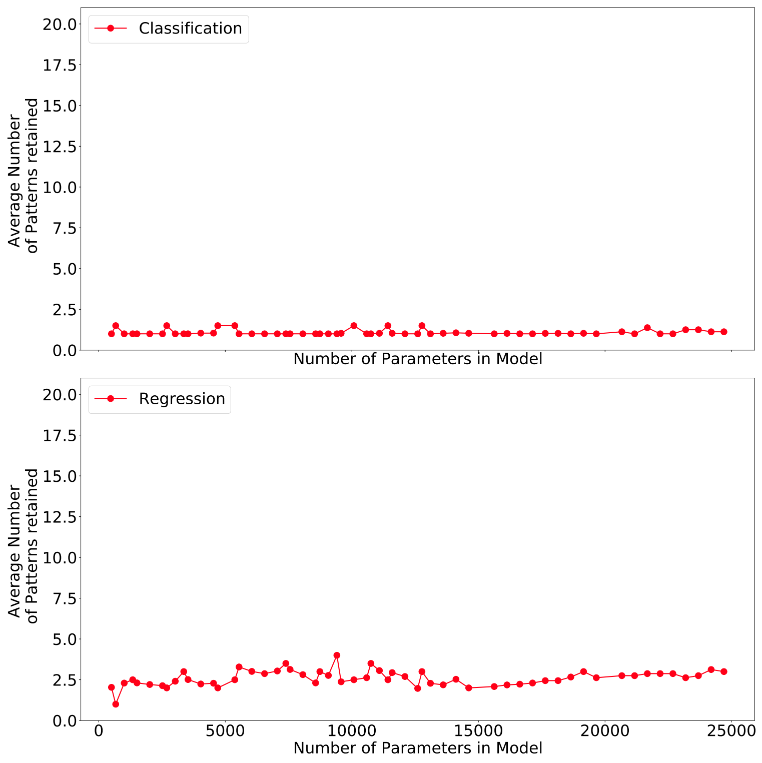

FIGURE 7.86: **Effect of the number of parameters on the ability to retain patterns over each activation function: tanh**

The observed relation between the activation function used by the RNNs and number of patterns retained per parameter is further analysed using table 7.12. Where table 7.12 captures the Spearman rank correlation coefficients for each activation function for each RNN and the number of patterns retained. The purpose of this table is to determine which activation function RNN pair would have the most benefit in increasing the number of parameters used.

Considering the regression experiment first, the most negatively correlated pairs are: bidirectional Elman-softplus, bidirectional GRU-hard sigmoid, bidirectional Jordan- softplus, bidirectional LSTM-hard sigmoid, Elman-elu, GRU-softmax, Jordan-softplus and the LSTM-sigmoid. Since the sigmoid activation function is used in the internal workings of the LSTM, the results observed indicates that replacing this activation function in the LSTM node itself may lead to better performance. However investigating the effects of replacing the internal activation functions of the LSTM is not within the scope of this dissertation. The most positively correlated to performance increase are: bidirectional

Elman-softsign, bidirectional GRU-softplus, bidirectional Jordan-softsign, bidirectional LSTM-softplus, Elman-softsign, GRU-softsign, Jordan-softsign and the LSTM-softplus.

The behaviour between the regression and classification experiments differ significantly. For the classification case the most positively correlated pairs are: bidirectional Elman-softsign, bidirectional GRU-relu, bidirectional Jordan-softsign, bidirectional LSTM-tanh, Elman-tanh, GRU-selu, Jordan-softsign and the LSTM-tanh. Thus it is clear that the only RNNs that have a consistent behaviour profile between these experiments are the bidirectional Elman, bidirectional Jordan and Jordan. This indicates that RNN behaviour is dependant on the problem formulation. Next the actual per parameter pattern capacity ratio will be inspected.

Figures 7.87, 7.88, 7.89, 7.90, 7.91, 7.92, 7.93, 7.94, 7.95 and 7.96 illustrate the per parameter pattern capacity ratio achieved on average for all neural networks using specific activation functions, in order to retain the most number of patterns for both classification and regression variants of the experiment. As seen previously, the same degradation in per parameter capacity ratio as discussed in section 7.2.1 is observed. In all cases the regression experiment ensured a larger per parameter capacity ratio than the classification experiment. To be more granular in the analysis, table 7.13 needs to be referenced as to gauge the average per parameter pattern capacity ratio for each RNN using different activation functions.

Activation Function		Bidirectional elamn	Bidirectional GRU	Bidirectional Jordan	Bidirectional LSTM	Elman	GRU	Jordan	LSTM
elu	Classification	-0.3313	0.3666	-0.2714	0.0000	0.0000	0.0000	0.0124	0.0000
	Regression	-0.5134	0.5349	-0.5305	0.2962	-0.5949	0.4343	-0.3937	0.2993
hard sigmoid	Classification	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	Regression	-0.3627	-0.4915	-0.1736	-0.4032	-0.2805	-0.4566	-0.4721	-0.3726
linear	Classification	0.0000	0.3627	0.0000	0.3859	0.2594	0.3267	0.2753	0.0000
	Regression	0.1366	0.5326	0.2048	0.3496	0.0270	0.4737	-0.1786	0.3062
relu	Classification	0.0000	0.4795	0.0000	0.0000	0.0000	0.0000	-0.3190	0.0000
	Regression	-0.3759	0.0503	-0.2298	0.1221	-0.4213	-0.2049	-0.3352	0.2815
selu	Classification	0.1330	0.3461	0.2196	0.0000	0.0000	0.3472	0.2777	0.0000
	Regression	-0.3889	0.2484	-0.2817	0.2442	-0.3094	0.2368	-0.1493	0.2367
sigmoid	Classification	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	Regression	-0.3758	-0.3847	-0.4873	-0.3014	-0.5428	-0.4929	-0.5364	-0.4091
softmax	Classification	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	Regression	0.0000	0.7354	0.0000	0.0000	-0.5843	-0.7139	-0.5324	0.0000
softplus	Classification	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	Regression	-0.9032	0.9047	-0.5932	0.9678	-0.0205	0.8526	-0.5682	0.9378
softsign	Classification	0.1997	0.3538	0.5963	0.0000	0.0000	0.3006	0.5546	0.0000
	Regression	0.9652	0.8633	0.9291	0.9509	0.9681	0.8967	0.9232	0.9205
tanh	Classification	-0.3019	0.4101	0.0000	0.3912	0.3419	0.2448	-0.2345	0.2067
	Regression	-0.3936	0.1789	-0.3076	0.3967	-0.5250	-0.0908	-0.4146	0.2424

TABLE 7.12: Spearman correlation between the number of patterns retained and the number of network parameters per activation function for each RNN

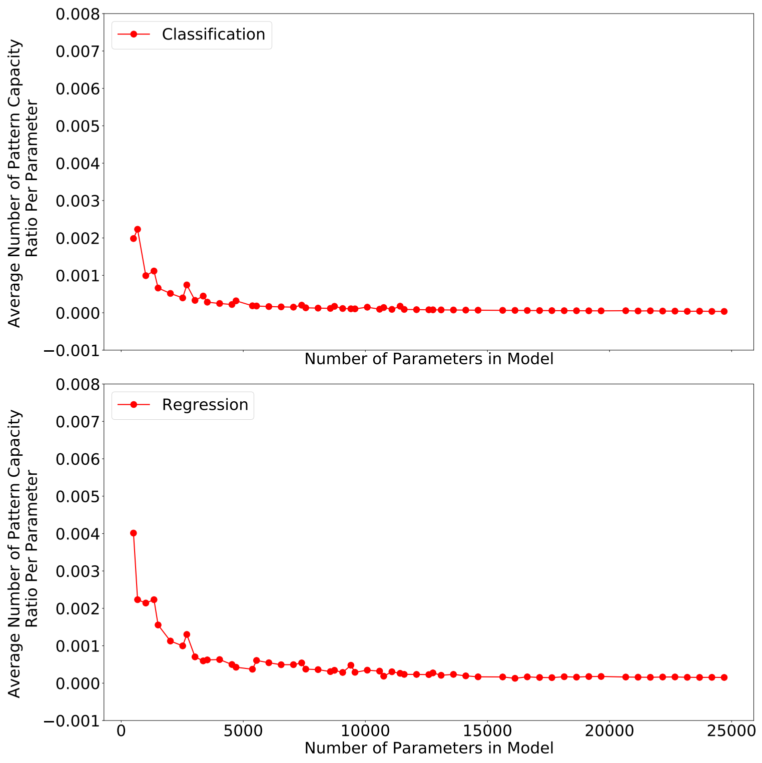

FIGURE 7.87: **Effect of increasing the number of parameters on the per parameter capacity of the network using elu activation function**

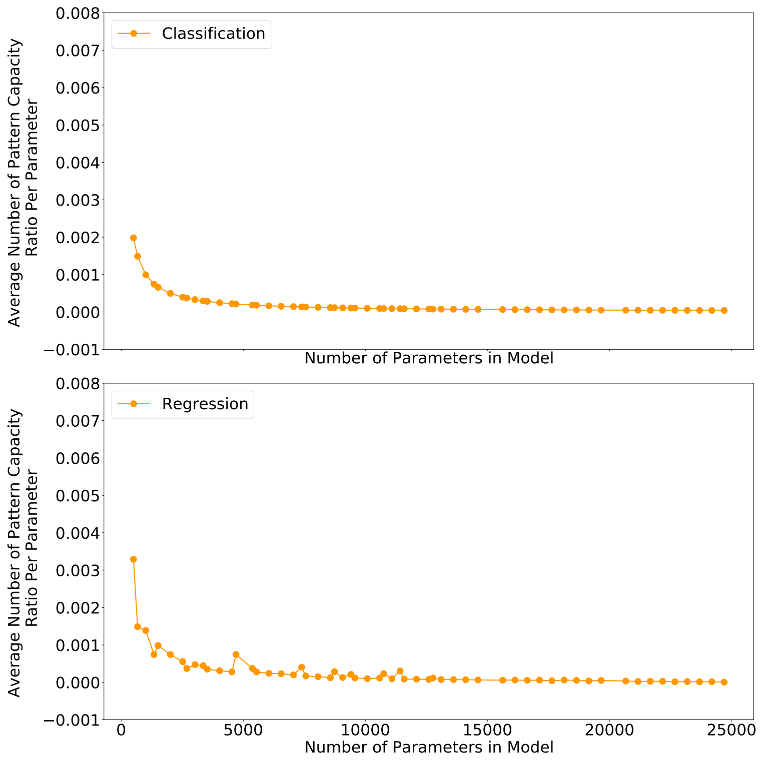

FIGURE 7.88: **Effect of increasing the number of parameters on the per parameter capacity of the network using hard sigmoid activation function**

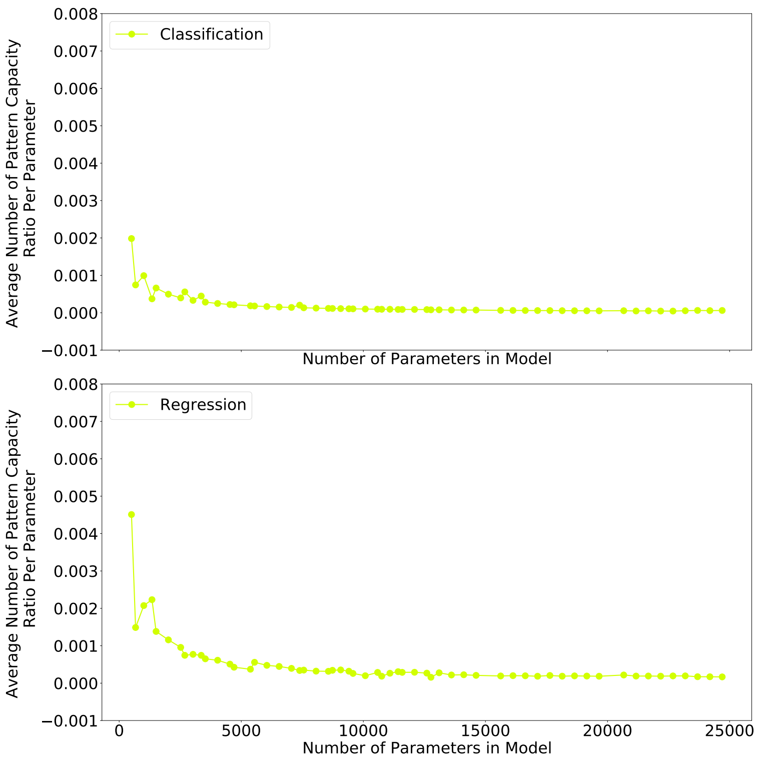

FIGURE 7.89: **Effect of increasing the number of parameters on the per parameter capacity of the network using linear activation function**

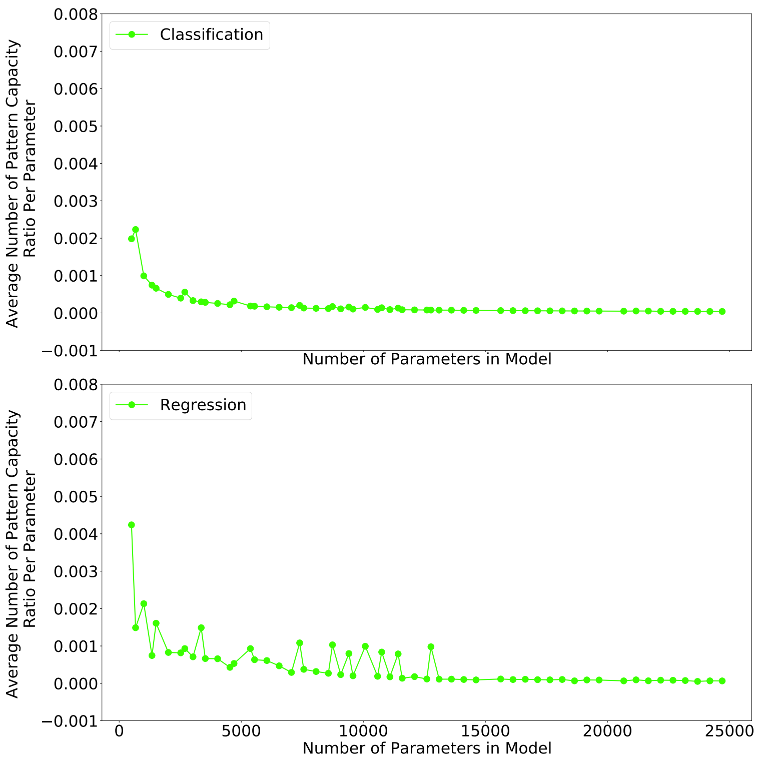

FIGURE 7.90: **Effect of increasing the number of parameters on the per parameter capacity of the network using relu activation function**

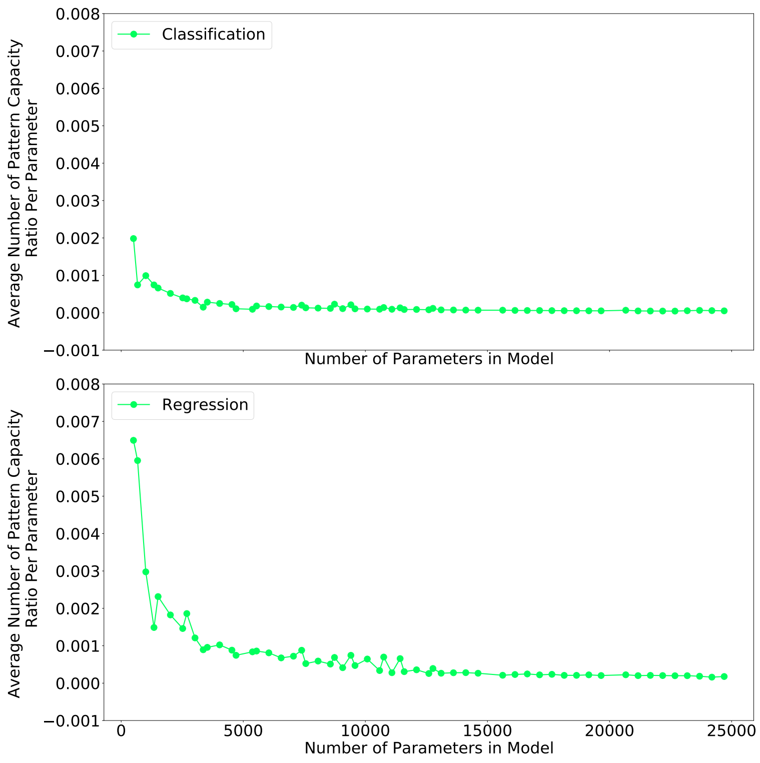

FIGURE 7.91: **Effect of increasing the number of parameters on the per parameter capacity of the network using selu activation function**

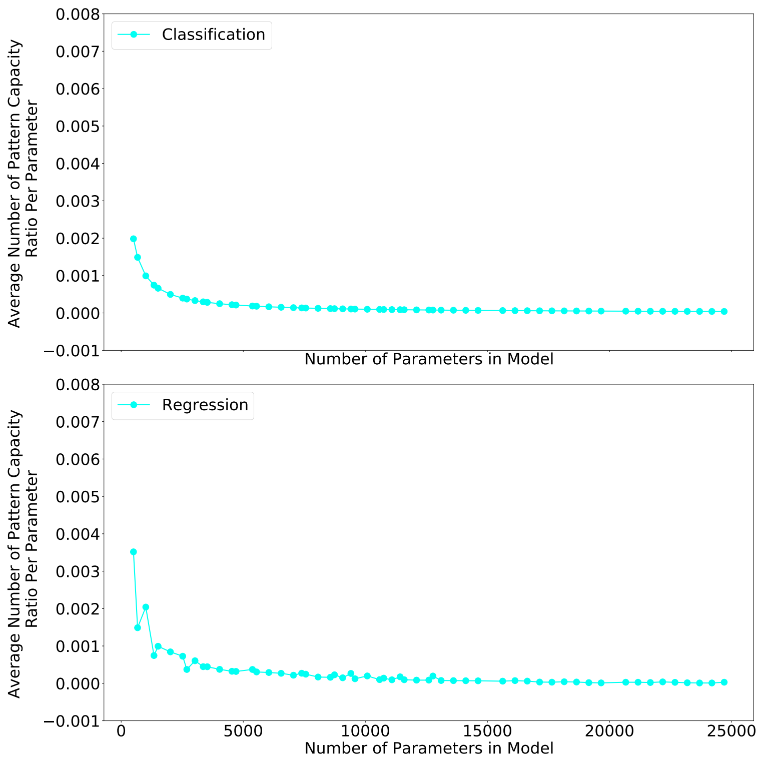

FIGURE 7.92: **Effect of increasing the number of parameters on the per parameter capacity of the network using sigmoid activation function**

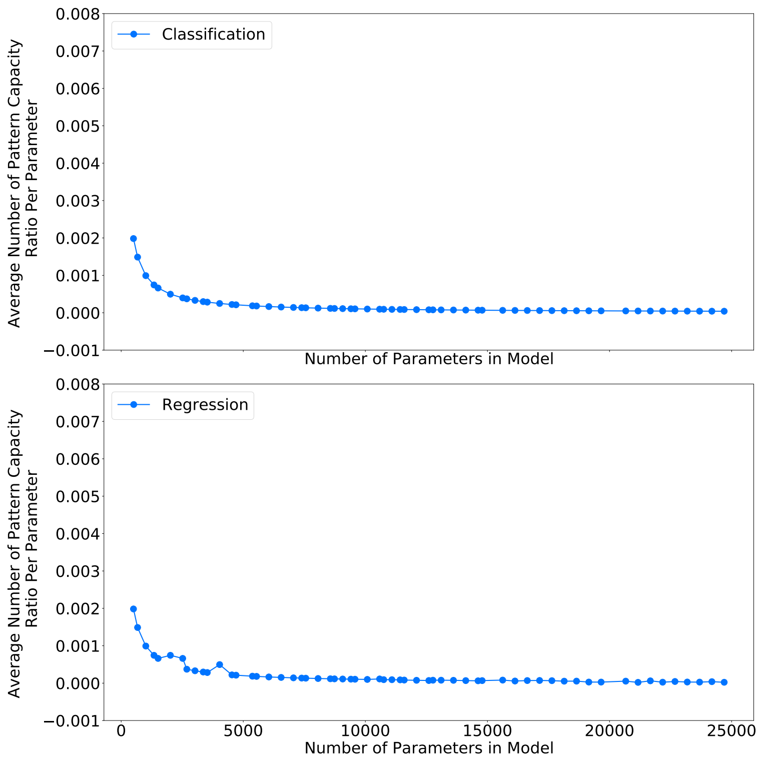

FIGURE 7.93: **Effect of increasing the number of parameters on the per parameter capacity of the network using softmax activation function**

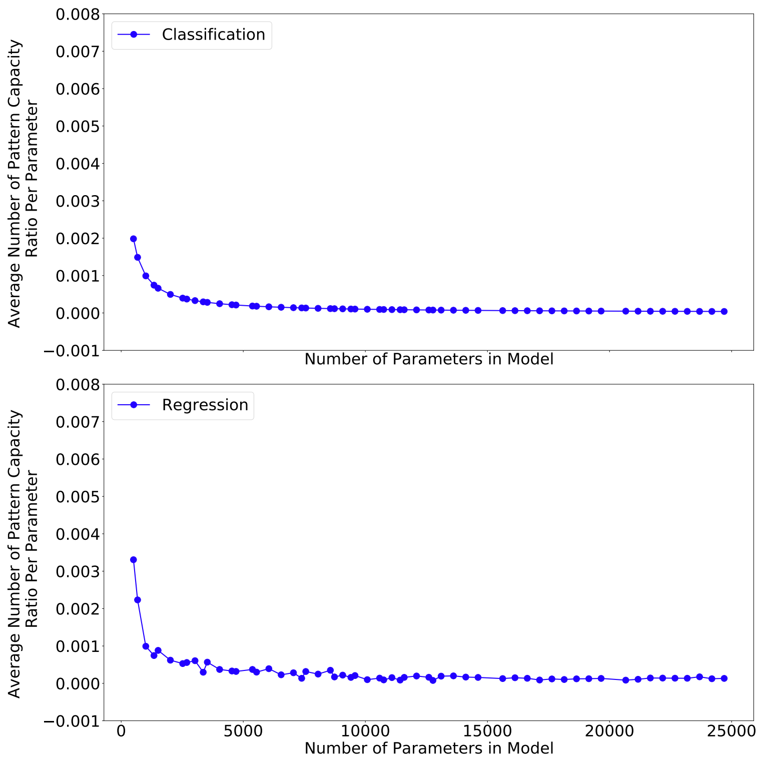

FIGURE 7.94: **Effect of increasing the number of parameters on the per parameter capacity of the network using softplus activation function**

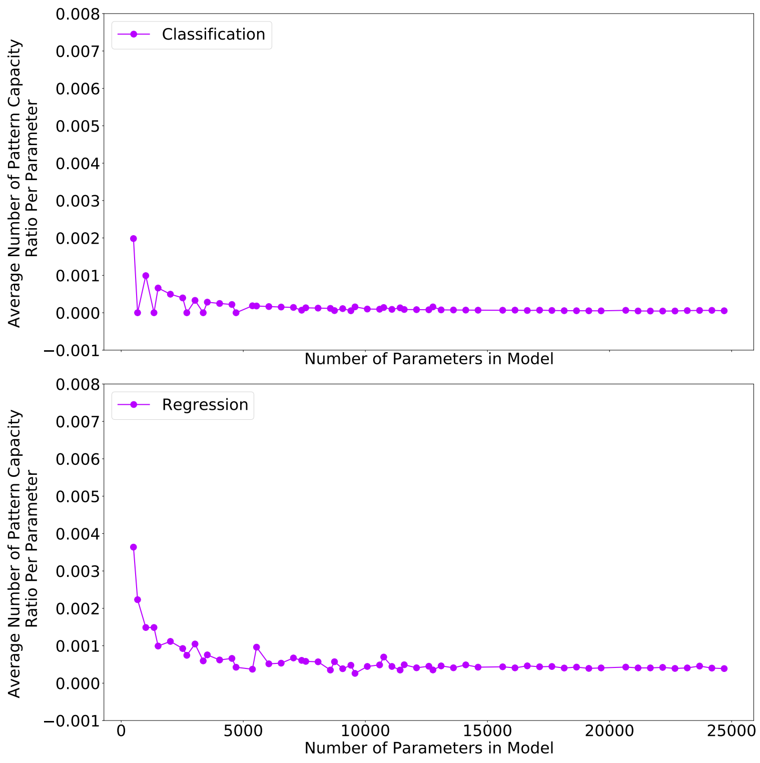

FIGURE 7.95: **Effect of increasing the number of parameters on the per parameter capacity of the network using softsign activation function**

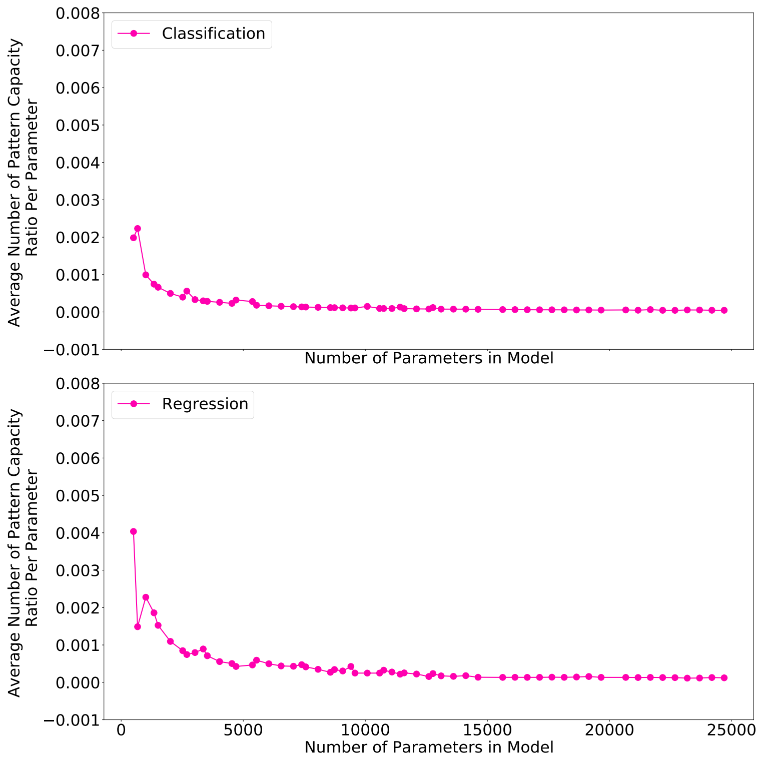

FIGURE 7.96: **Effect of increasing the number of parameters on the per parameter capacity of the network using tanh activation function**

Table 7.13 reports the average per parameter pattern capacity achieved by each activation function applied to each RNN. Upon inspection of the regression versus the classification formulation of the experiment, it is clear that in all cases, the per parameter capacity is larger for the regression experiment. This corresponds to findings made in the sequence length retention as discussed in section 7.1.2 which alludes to RNNs responding better to regression problems than the classification counterparts. However, the number of experiments is too small to make any true conclusions. The observed behaviour of RNNs being more applicable to regression than classification problem statements may be attributed to how the classification problem was formulated or it may be due to the increase in parameters in the network. The discrepancy in performance of RNNs in the regression compared to classification variants of experiments warrants further investigation, however the investigation thereof is outside of the scope of this dissertation.

Evaluating the effectiveness of the RNNs, using different activation functions, it is clear that the most effective per parameter pattern capacity is achieved by the LSTM RNN using the selu activation function followed by the bidirectional LSTM also using selu. However the difference between the per parameter pattern capacity is only marginal. Despite this, the activation function which achieved the highest per parameter pattern capacity over all the RNNs was the softplus. The softplus activation function has a large range $(0, +\infty)$ and a smooth slope which might make it more versatile to use across various RNNs, however further investigation is required to validate this statement.

7.2.3 Relationship between the Number of Patterns and Number Of Layers

In this section the relationship between the number of patterns retained and the number of hidden layers used in the network will be investigated.

The average number of patterns retained per additional hidden layer for each RNN is presented in figures 7.97, 7.98, 7.99, 7.100, 7.101, 7.102, 7.103 and 7.104. From these figures it is clear that in the classification experiment, increasing the number of hidden layers did not improve the performance of any RNN. In the regression experiment the

Activation Function		Bidirectional Elman	Bidirectional GRU	Bidirectional Jordan	Bidirectional LSTM	Elman	GRU	Jordan	LSTM
elu	Classification	0.00010	0.00011	0.00010	0.00011	0.00011	0.00011	0.00010	0.00010
	Regression	0.00015	0.00048	0.00013	0.00037	0.00014	0.00046	0.00013	0.00043
hard sigmoid	Classification	0.00010	0.00011	0.00010	0.00011	0.00011	0.00011	0.00010	0.00011
	Regression	0.00011	0.00012	0.00011	0.00012	0.00011	0.00013	0.00009	0.00014
linear	Classification	0.00010	0.00012	0.00010	0.00011	0.00011	0.00011	0.00010	0.00011
	Regression	0.00023	0.00042	0.00020	0.00038	0.00024	0.00039	0.00018	0.00038
relu	Classification	0.00010	0.00012	0.00010	0.00011	0.00011	0.00011	0.00010	0.00011
	Regression	0.00017	0.00024	0.00019	0.00023	0.00020	0.00025	0.00016	0.00029
selu	Classification	0.00010	0.00011	0.00010	0.00011	0.00011	0.00011	0.00010	0.00011
	Regression	0.00023	0.00049	0.00021	0.00063	0.00028	0.00051	0.00025	0.00065
sigmoid	Classification	0.00010	0.00011	0.00010	0.00011	0.00011	0.00011	0.00010	0.00011
	Regression	0.00011	0.00016	0.00010	0.00016	0.00011	0.00015	0.00010	0.00015
softmax	Classification	0.00007	0.00007	0.00007	0.00008	0.00008	0.00007	0.00007	0.00008
	Regression	0.00007	0.00018	0.00007	0.00007	0.00002	0.00002	0.00005	0.00007
softplus	Classification	0.00007	0.00007	0.00007	0.00007	0.00008	0.00008	0.00007	0.00008
	Regression	-0.00005	0.00024	0.00006	0.00036	0.00010	0.00023	0.00008	0.00025
softsign	Classification	0.00008	0.00008	0.00007	0.00008	0.00008	0.00009	0.00008	0.00008
	Regression	0.00051	0.00041	0.00042	0.00047	0.00043	0.00041	0.00042	0.00051
tanh	Classification	0.00010	0.00012	0.00010	0.00011	0.00011	0.00011	0.00010	0.00011
	Regression	0.00013	0.00043	0.00013	0.00050	0.00013	0.00031	0.00011	0.00035

Table 7.13: Per parameter capacity between the number of patterns retained and the number of network parameters per activation function for each RNN

picture changes. The average pattern retention reduced for the bidirectional Elman, bidirectional Jordan, the Elman and Jordan RNNs as more hidden layers were added. In the case of the bidirectional GRU, the bidirectional LSTM, GRU and LSTM RNNs a pattern can be observed. The number of patterns retained increases as the number of layers do. However between the increase from two hidden layers to three, there seems to be a decrease or no increase in retention on average, which is rather peculiar and may be an artifact of the experiment itself. The behaviour of no increase or even a decrease in retention when increasing the number of layers from two to three, would be an ideal candidate for further research. To investigate further behavioural traits of how the RNNs react to an increasing number of layers, an analysis of table 7.14 is done next.

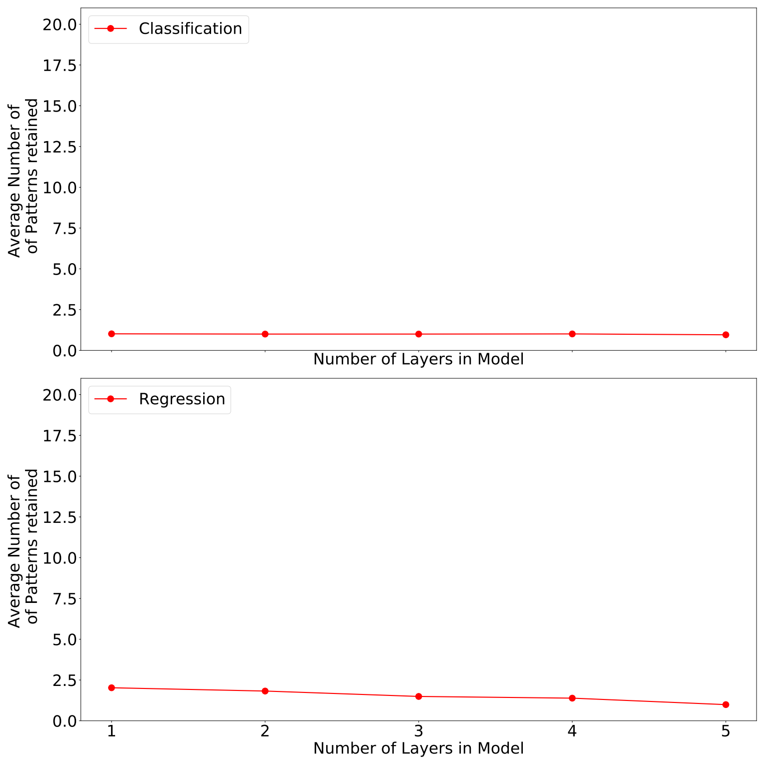

FIGURE 7.97: **Effect of increasing number of layers the on the ability to retain patterns for the bidirectional Elman RNN**

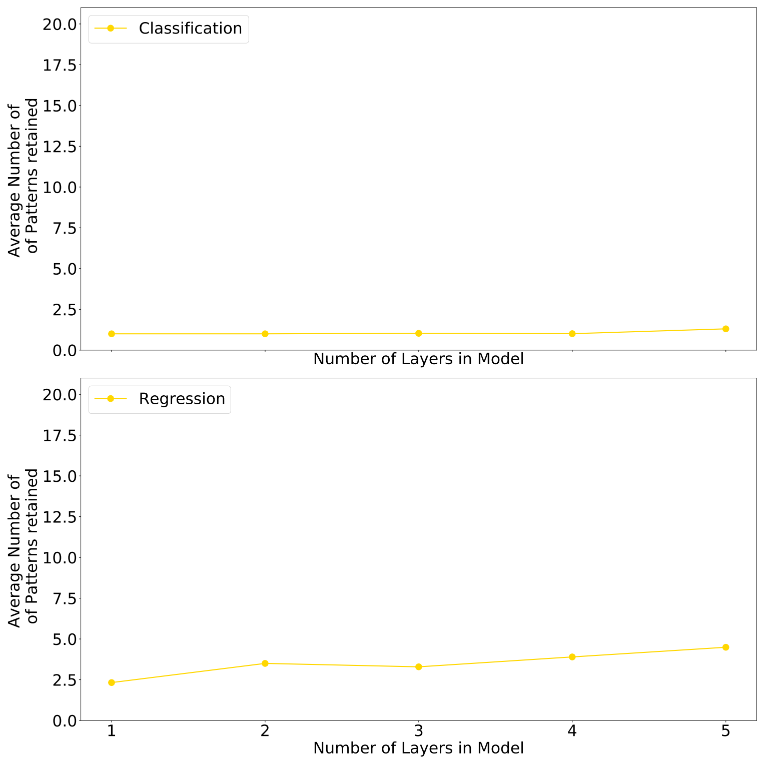

Figure 7.98: **Effect of increasing number of layers the on the ability to retain patterns for the bidirectional GRU RNN**

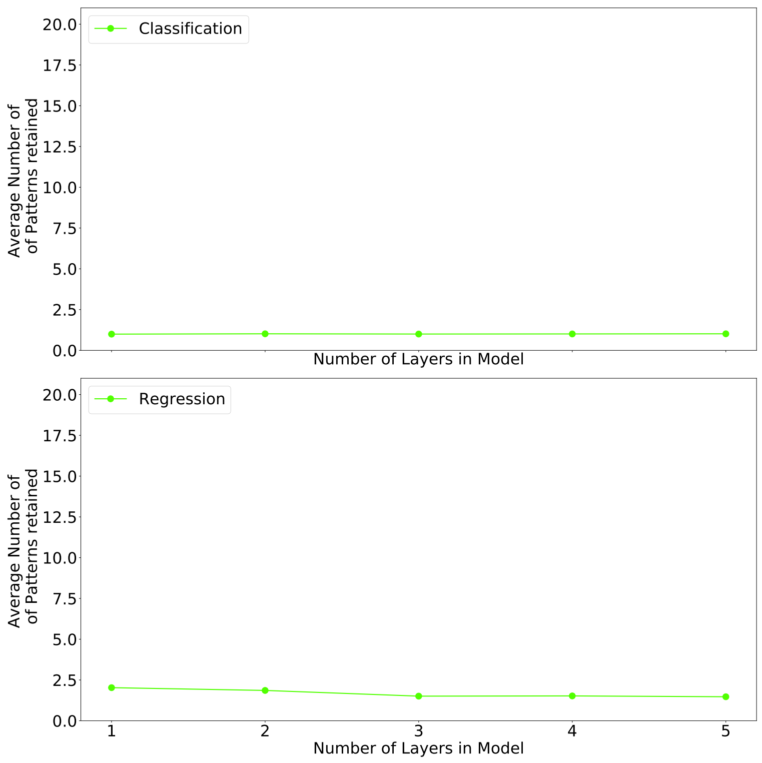

Figure 7.99: **Effect of increasing number of layers the on the ability to retain patterns for the bidirectional Jordan RNN**

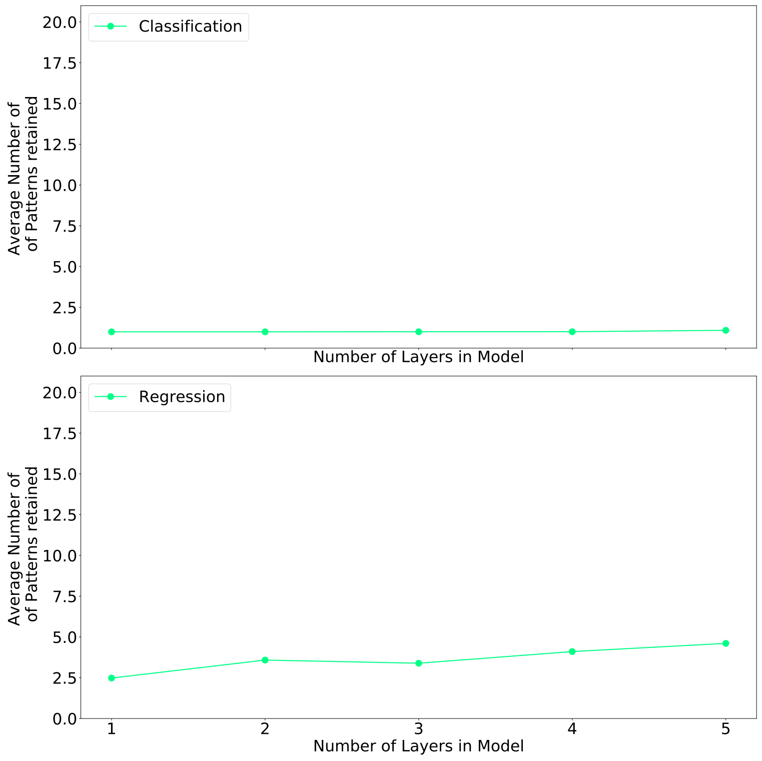

FIGURE 7.100: **Effect of increasing number of layers the on the ability to retain patterns for the bidirectional LSTM RNN**

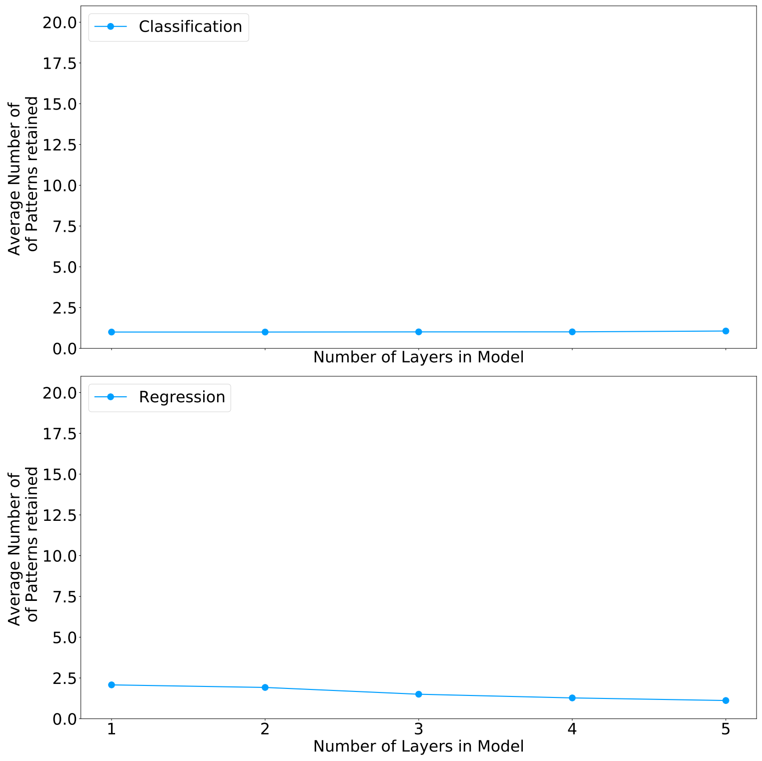

FIGURE 7.101: **Effect of increasing number of layers the on the ability to retain patterns for the Elman RNN**

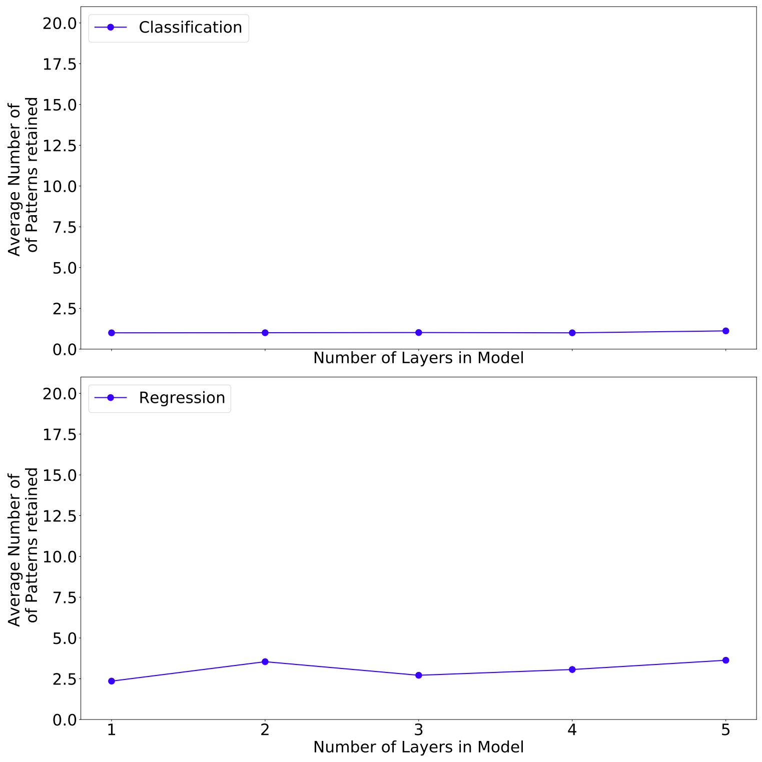

FIGURE 7.102: **Effect of increasing number of layers the on the ability to retain patterns for the GRU RNN**

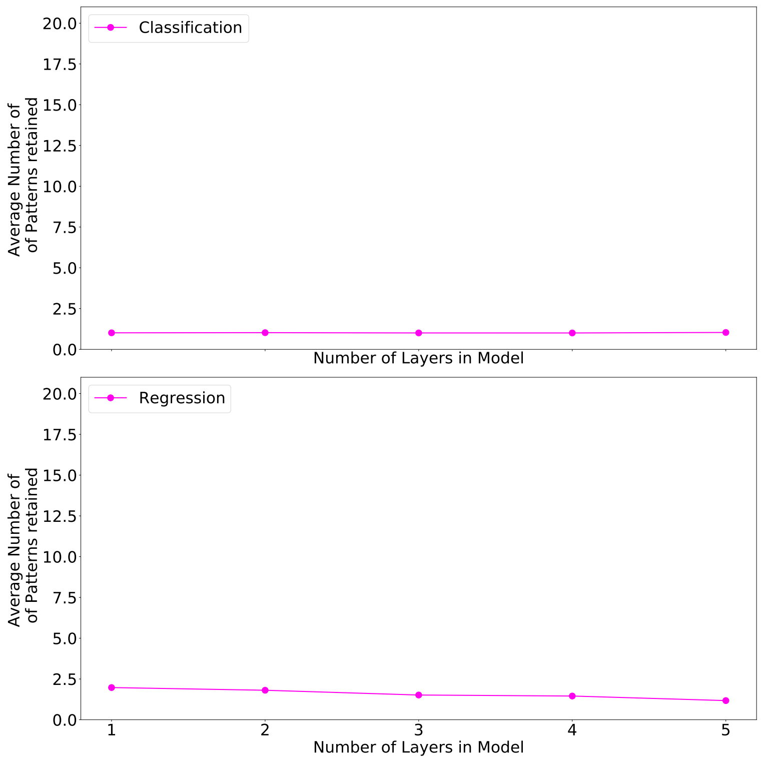

FIGURE 7.103: **Effect of increasing number of layers the on the ability to retain patterns for the Jordan RNN**

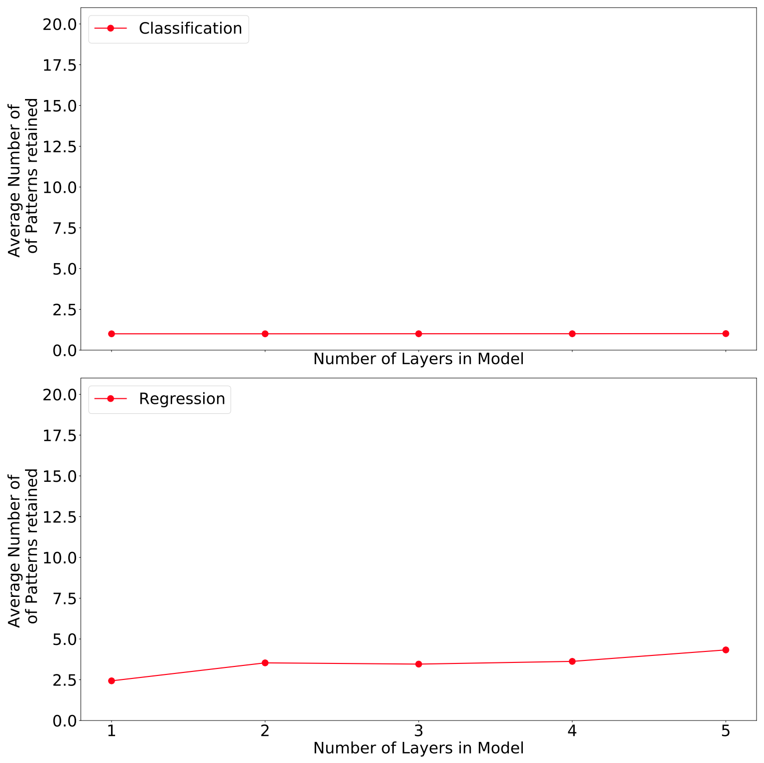

FIGURE 7.104: **Effect of increasing number of layers the on the ability to retain patterns for the LSTM RNN**

The Spearman rank correlation between the number of patterns retained and the number of hidden layers is captured in table 7.14. From this table it can be observed that the behaviour of the RNNs between the classification and regression cases do not align as discussed when evaluating figures 7.97, 7.98, 7.99, 7.100, 7.101, 7.102, 7.103 and 7.104. However, this table does also indicate that the relationship observed between the transition between two to three hidden layers exists. On average the RNN which is the most positively correlated to a performance increase when increasing the number of hidden layers is the GRU in the classification experiment and the bidirectional Jordan for the regression experiment. To determine which RNNs are the most effective in retaining patterns, an evaluation of the per parameter pattern capacity will be performed next.

Figures 7.105, 7.106, 7.107, 7.108, 7.109, 7.110, 7.111 and 7.112 illustrate the per parameter pattern capacity for an increasing number of layers. Here it is clear that the per parameter capacity ratio decreases as more layers are added. The largest of which occurs when increasing from one to two layers in both the classification and regression experiments. A more granular analysis will be done using table 7.15.

RNN		Number of layers					
		1	2	3	4	5	Average
Bidirectional Elman	Classification	0.0326	0.0000	0.0000	-0.0836	-0.1340	-0.03700
	Regression	0.0072	-0.2533	-0.0114	-0.0956	-0.0463	-0.07988
Bidirectional GRU	Classification	0.0000	0.0000	-0.0293	-0.1257	0.0826	-0.01448
	Regression	0.0791	-0.1173	0.0015	0.1208	0.0400	0.02482
Bidirectional Jordan	Classification	0.0014	0.0843	0.0000	0.0211	-0.0983	0.00170
	Regression	-0.0052	0.1997	0.0459	-0.0201	-0.0472	0.03462
Bidirectional LSTM	Classification	0.0000	0.0000	0.0952	0.0638	0.0623	0.04426
	Regression	0.0344	-0.0528	0.0111	0.0374	0.0465	0.01532
Elman	Classification	0.0000	0.0000	0.0139	-0.0570	0.0028	-0.00806
	Regression	0.0890	-0.1893	0.0530	-0.1205	-0.0033	-0.03422
GRU	Classification	0.0000	0.1128	-0.0075	0.0000	0.3111	0.08328
	Regression	0.0401	-0.0929	0.0174	-0.0084	-0.0034	-0.00944
Jordan	Classification	0.0726	0.1660	0.0595	-0.0211	0.1437	0.08414
	Regression	0.0119	0.2086	0.0983	-0.1144	-0.0437	0.03214
LSTM	Classification	0.0000	0.0000	0.0651	0.0192	0.1368	0.04422
	Regression	0.0283	0.0108	-0.0598	0.1057	0.0395	0.02490

TABLE 7.14: Spearman correlation between the number of patterns retained and the number of network parameters for increasing NN depth for each RNN

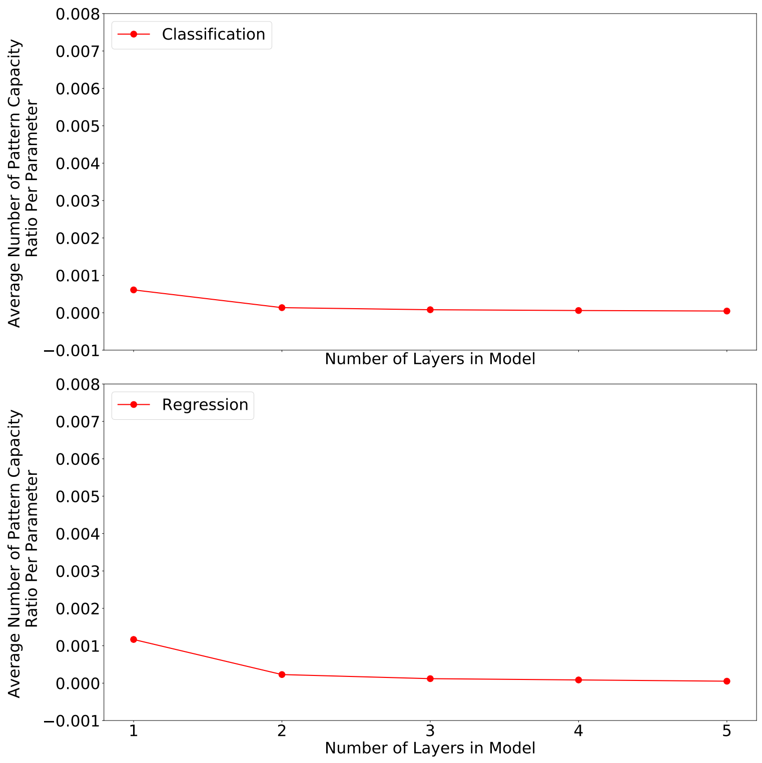

FIGURE 7.105: **Effect of increasing the number of parameters on the per parameter capacity of the network when increasing the number of layers for the Jordan RNN**

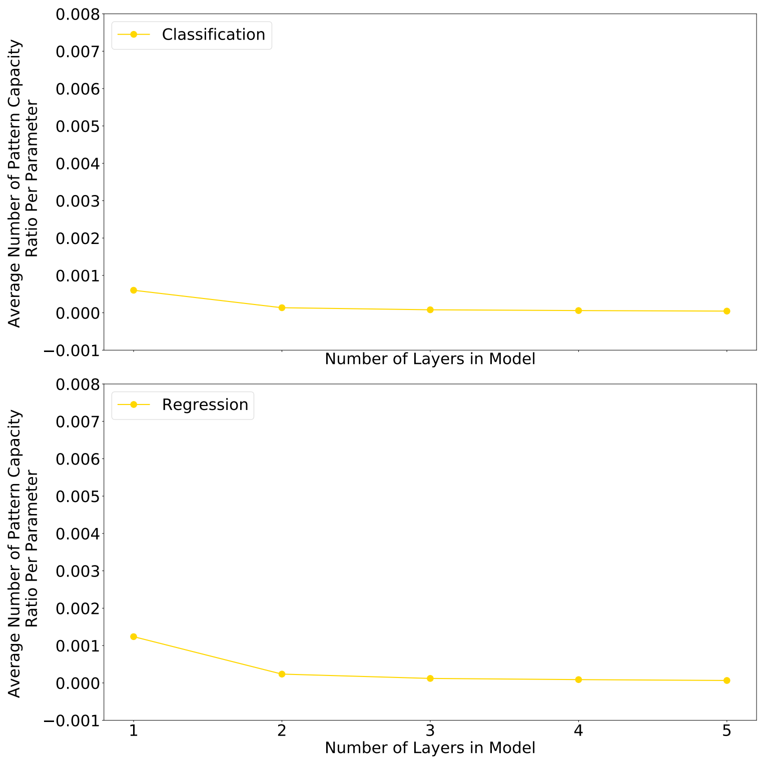

FIGURE 7.106: **Effect of increasing the number of parameters on the per parameter capacity of the network when increasing the number of layers for the bidirectional Jordan RNN**

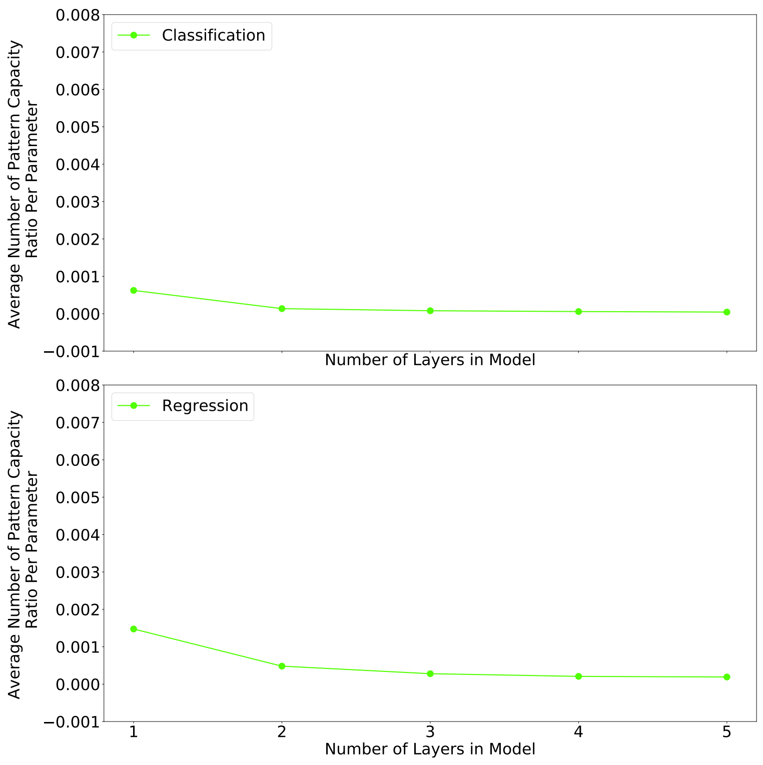

FIGURE 7.107: **Effect of increasing the number of parameters on the per parameter capacity of the network when increasing the number of layers for the LSTM RNN**

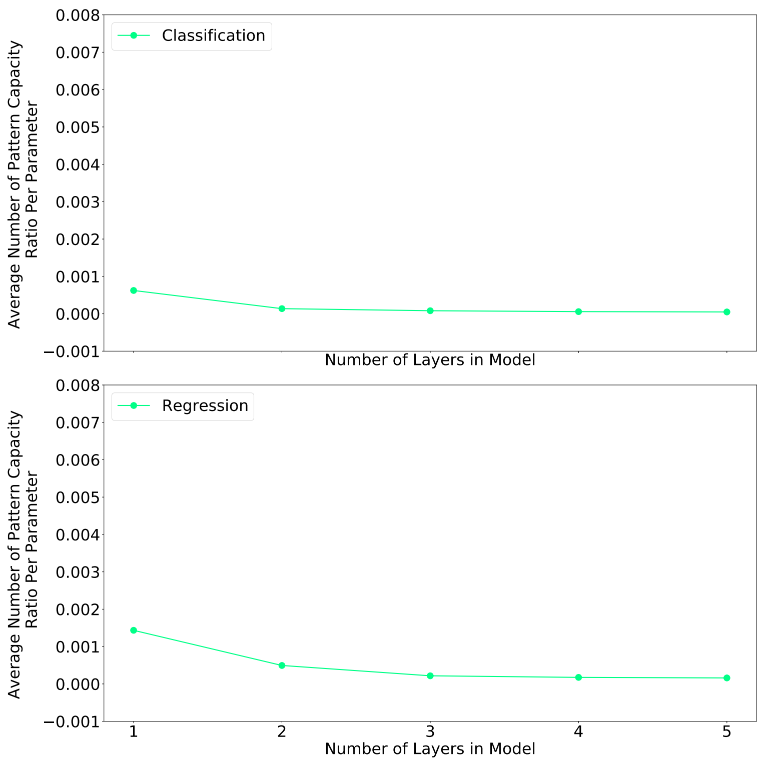

FIGURE 7.108: **Effect of increasing the number of parameters on the per parameter capacity of the network when increasing the number of layers for the GRU RNN**

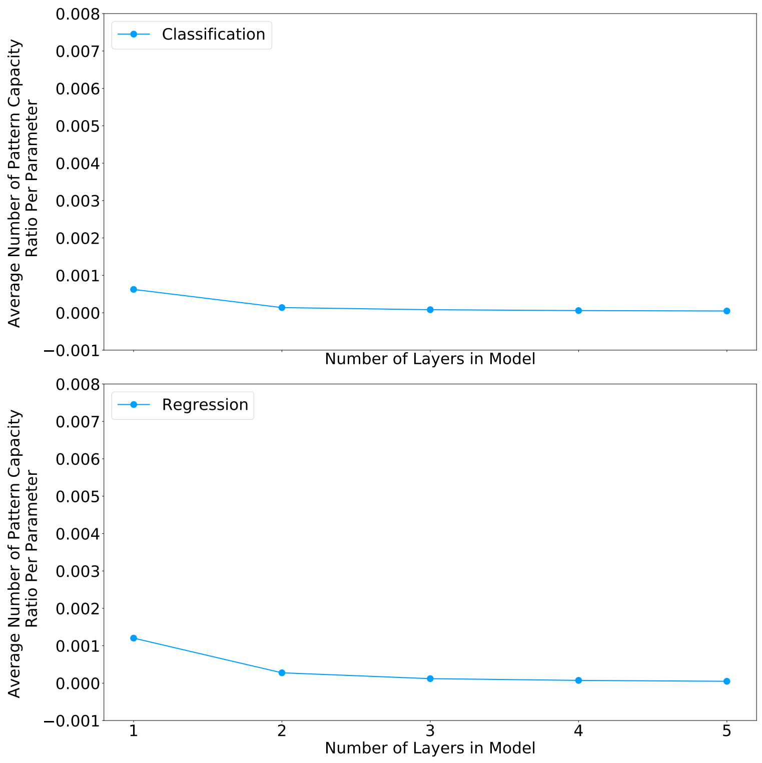

FIGURE 7.109: **Effect of increasing the number of parameters on the per parameter capacity of the network when increasing the number of layers for the Elman RNN**

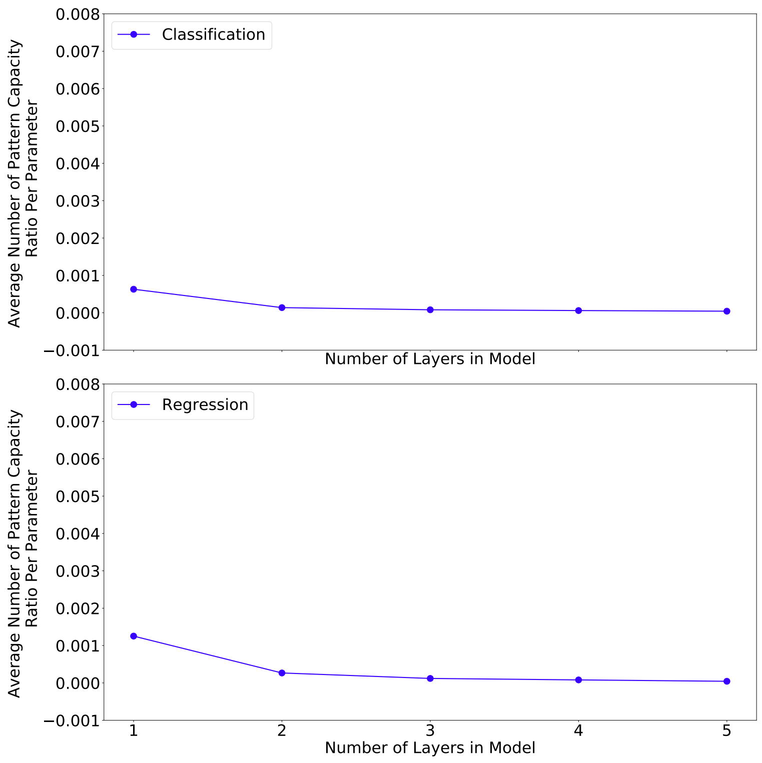

FIGURE 7.110: **Effect of increasing the number of parameters on the per parameter capacity of the network when increasing the number of layers for the bidirectional Elman RNN**

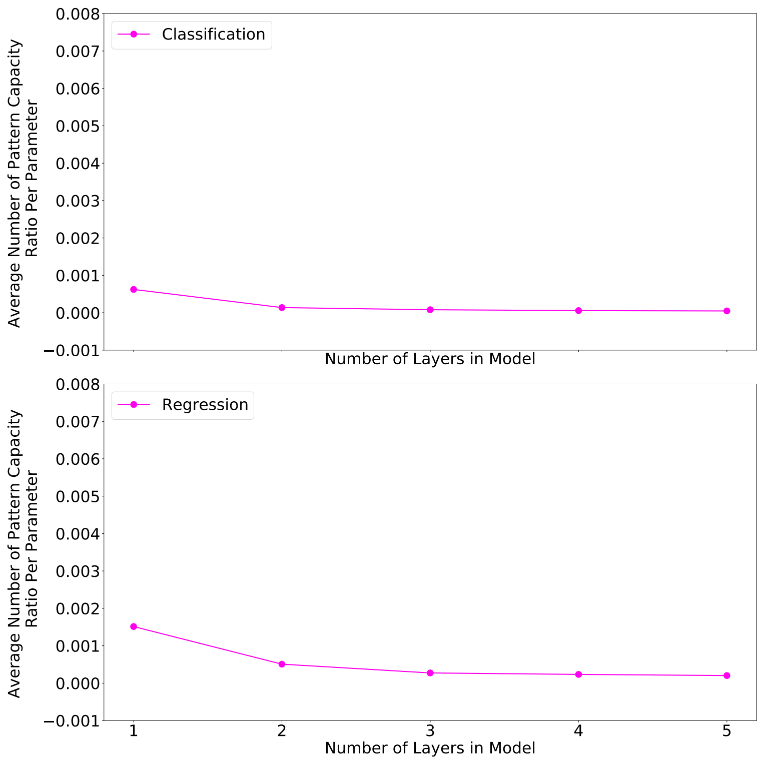

FIGURE 7.111: **Effect of increasing the number of parameters on the per parameter capacity of the network when increasing the number of layers for the bidirectional LSTM RNN**

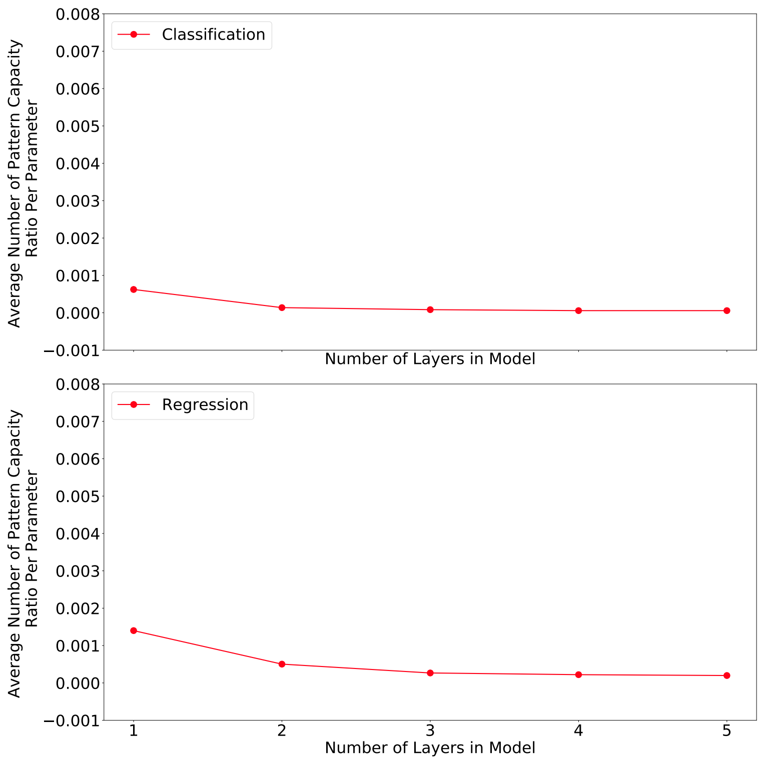

FIGURE 7.112: **Effect of increasing the number of parameters on the per parameter capacity of the network when increasing the number of layers for the bidirectional GRU RNN**

The results of the per parameter pattern capacity ratio as the number of hidden layers are increased for each RNN, is reported in table 7.15. The configuration that lead to the highest per parameter pattern retention was the bidirectional LSTM using one hidden layer at 0.00098 in the regression case, followed by the LSTM RNN at 0.00098 in the regression case. In the classification the highest per parameter pattern retention was achieved by three RNNs; the bidirectional Elman, Elman RNN and GRU RNN, achieving a ratio of 0.00040. The bidirectional LSTM also performed the best as the number of layers increased from one to five which attained a parameter pattern retention ratio of 0.00043 on average in the regression case, indicating that for deeper networks, bidirectional LSTM are more effective at retaining a larger array of patterns per parameter used. In the classification case, on average all RNNs achieved the same average parameter pattern retention ratio of 0.00014 which indicates that the bidirectional layers did not increase the effectiveness in retention ratios on average in this experiment. Since the bidirectional layers did not increase the effectiveness of the RNNs in the classification cases, but did increase the parameter pattern retention ratios in the regression case, it indicates that the classification formulation may be interfering with the performance of the bidirectional layers. The interference of the classification formulation on the performance of bidirectional layers may be attributed to loss of information in the latter layers as the number of parameters needs to be consistent across all experiments.

7.2.4 Relationship between the Number of Patterns and Different Training Algorithms

This section investigates the effects of using different training algorithms on the per parameter pattern retention of various RNNs.

Figures 7.113, 7.114, 7.115 and 7.116 show the average number of patterns retained for increasing the number of model parameters. Behavioural traits observed previously, are also observed here. These training algorithms are more effective in the case of regression as opposed to the classification experiment. Here it is clear that the Adam and RPROP training algorithms resulted in higher retention of patterns. However, this ability decreases as the number of model parameters increases. Interestingly enough the

RNN		Number of Layers					
		1	2	3	4	5	Average
Bidirectional Elman	Classification	0.00040	0.00013	0.00008	0.00006	0.00004	0.00014
	Regression	0.00080	0.00025	0.00012	0.00008	0.00004	0.00026
Bidirectional GRU	Classification	0.00039	0.00013	0.00008	0.00006	0.00006	0.00014
	Regression	0.00092	0.00047	0.00026	0.00022	0.00020	0.00041
Bidirectional Jordan	Classification	0.00038	0.00013	0.00008	0.00006	0.00004	0.00014
	Regression	0.00076	0.00023	0.00012	0.00009	0.00006	0.00025
Bidirectional LSTM	Classification	0.00039	0.00013	0.00008	0.00006	0.00005	0.00014
	Regression	0.00098	0.00049	0.00027	0.00023	0.00020	0.00043
Elman	Classification	0.00040	0.00013	0.00008	0.00006	0.00005	0.00014
	Regression	0.00082	0.00026	0.00012	0.00007	0.00005	0.00027
GRU	Classification	0.00040	0.00013	0.00008	0.00006	0.00005	0.00014
	Regression	0.00093	0.00047	0.00022	0.00017	0.00016	0.00039
Jordan	Classification	0.00039	0.00013	0.00008	0.00006	0.00005	0.00014
	Regression	0.00074	0.00023	0.00012	0.00008	0.00005	0.00024
LSTM	Classification	0.00039	0.00013	0.00008	0.00006	0.00004	0.00014
	Regression	0.00096	0.00047	0.00027	0.00021	0.00019	0.00042

Table 7.15: Per parameter capacity for each RNN per activation function at various layer depths

SGD and Nesterov enable the RNNs to retain a larger number of patterns the more parameters are used in the RNNs. To investigate the behaviour of SGD and Nesterov increasing effectiveness of pattern retention further, refer to table 7.16.

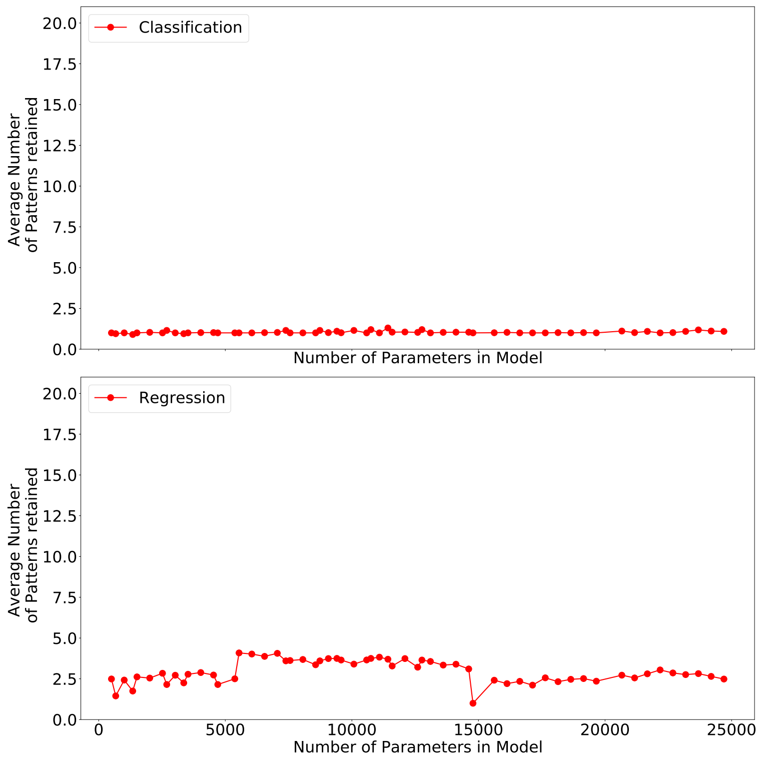

FIGURE 7.113: **Effect of using Adam training algorithm on the ability to retain patterns**

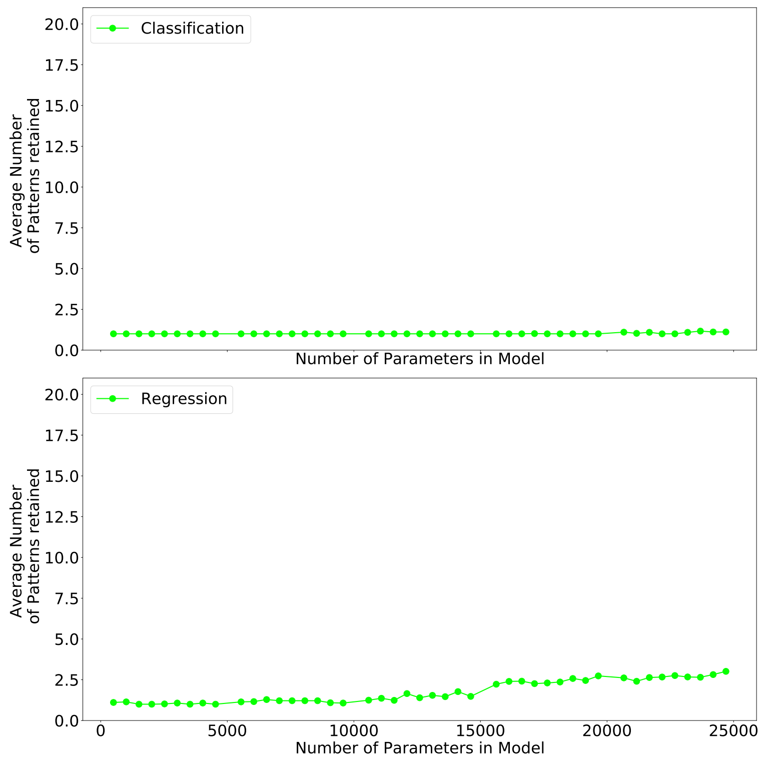

FIGURE 7.114: **Effect of using Nesterov training algorithm on the ability to retain patterns**

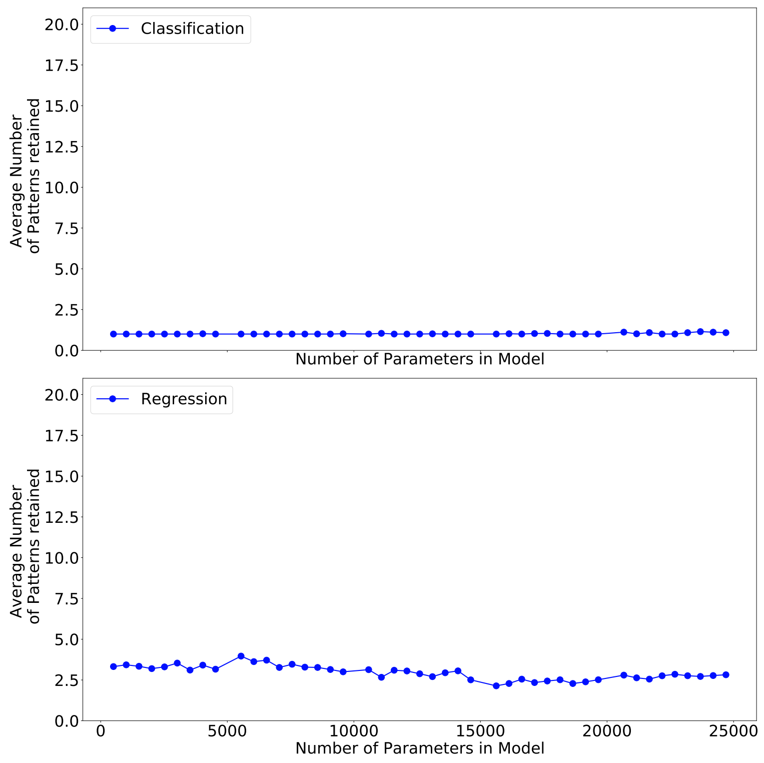

FIGURE 7.115: **Effect of using RPROP training algorithm on the ability to retain patterns**

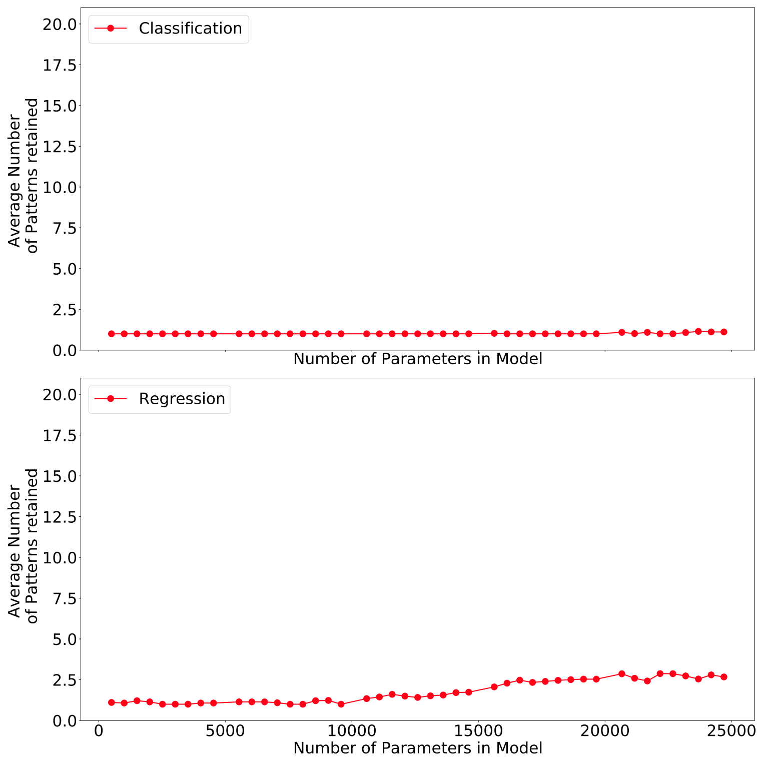

FIGURE 7.116: **Effect of using SGD training algorithm on the ability to retain patterns**

The Spearman rank correlation for each RNN using different training algorithms and the average number of patterns retained is presented in table 7.16 for both the regression and classification experiments. In general, the correlation between the number of patterns retained and the number of parameters in the model has less variance within the classification experiments, whereas this relationship is more variate during the regression experiments for all training algorithms utilised.

The highest correlation coefficient between the number of patterns retained and the number of parameters in the RNNs, was observed for the bidirectional GRU using the SGD training algorithm followed again by the same RNN using the Nesterov training algorithm during the regression experiment. Since the highest Spearman correlation coefficient was achieved by bidirectional GRU using the SGD and Nesterov training algorithms it is indicative of these training algorithms being more linear in their performance contribution as the number of parameters increase in the network. For the classification experiment, the most positive correlation coefficient of 0.6044 was also achieved by the bidirectional GRU using SGD. The most negatively correlated pair is the Elman RNN using RPROP in the case of regression with a correlation coefficient of -0.3376. For the classification experiment the bidirectional Elman was the most negatively correlated when using SGD to train it, with a correlation coefficient of -0.1328.

Figures 7.117, 7.118, 7.119 and 7.120 serve to show how per parameter pattern capacity decreases as the number of parameters in the RNNs increase. The impact of the training algorithms on the per parameter capacity ratio seem consistent across the classification experiment. During the regression experiment, the degradation happens more gradually when using the RPROP training algorithm. Here the Adam training algorithm performs well initially, but the retention rate drops sharply. This will further be explored in table 7.17.

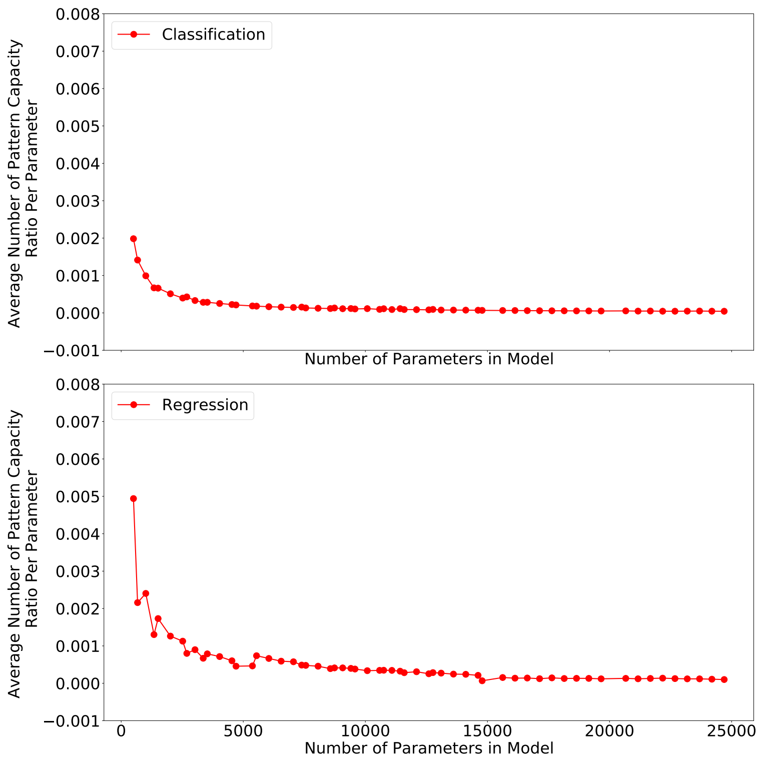

FIGURE 7.117: **Effect of increasing the number of parameters on the per parameter capacity of the network using Adam training algorithm**

		Training Algorithms			
RNN		Adam	Nesterov	RPROP	SGD
Bidirectional Elman	Classification	-0.1328	-0.0890	-0.0399	-0.0453
	Regression	-0.3061	-0.0042	-0.2528	0.0344
Bidirectional GRU	Classification	0.3054	0.3329	0.2921	0.3335
	Regression	0.2121	0.5941	-0.0083	0.6044
Bidirectional Jordan	Classification	0.0385	0.0547	0.0357	-0.0040
	Regression	-0.1952	0.1251	-0.2757	0.0943
Bidirectional LSTM	Classification	0.2031	0.1774	0.1814	0.1814
	Regression	0.1764	0.5626	0.0434	0.5469
Elman	Classification	0.1445	0.1456	0.1553	0.1485
	Regression	-0.2736	0.0144	-0.3376	0.0540
GRU	Classification	0.1898	0.2226	0.1600	0.2223
	Regression	0.1121	0.4898	-0.1464	0.4810
Jordan	Classification	0.0004	0.0654	0.0322	0.0387
	Regression	-0.2048	0.0516	-0.3223	0.0223
LSTM	Classification	0.0737	0.0708	0.0837	0.0700
	Regression	0.1753	0.5465	-0.0224	0.5515

Table 7.16: Spearman correlation between the number of patterns retained and the number of network parameters using different training algorithms for each RNN

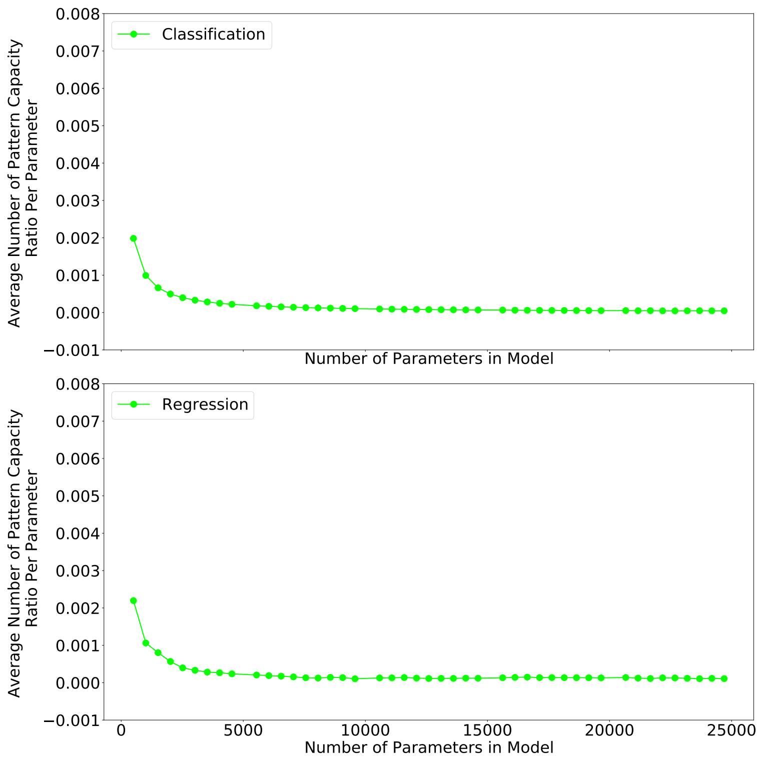

FIGURE 7.118: **Effect of increasing the number of parameters on the per parameter capacity of the network using SGD training algorithm**

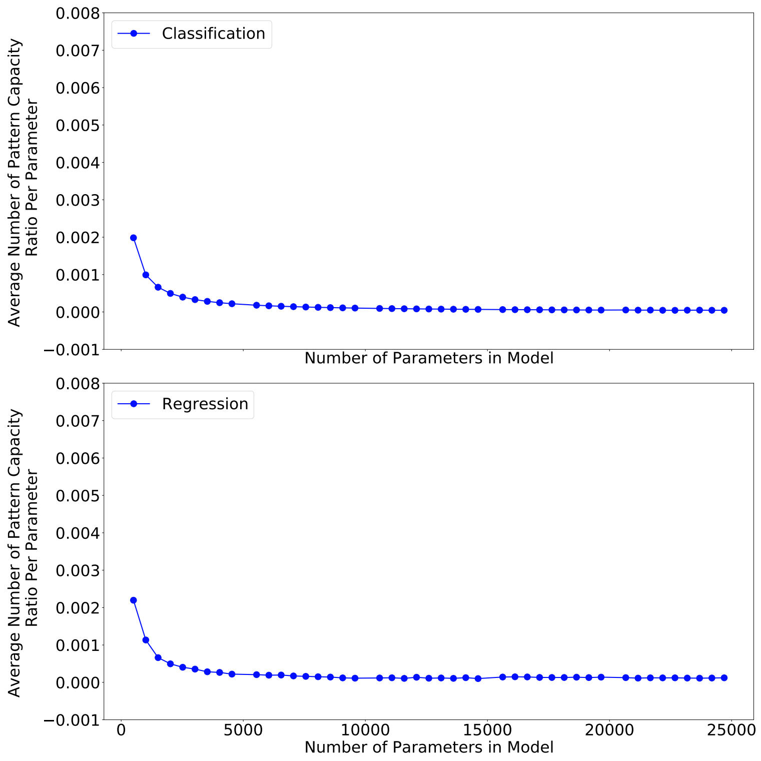

FIGURE 7.119: **Effect of increasing the number of parameters on the per parameter capacity of the network using Nesterov training algorithm**

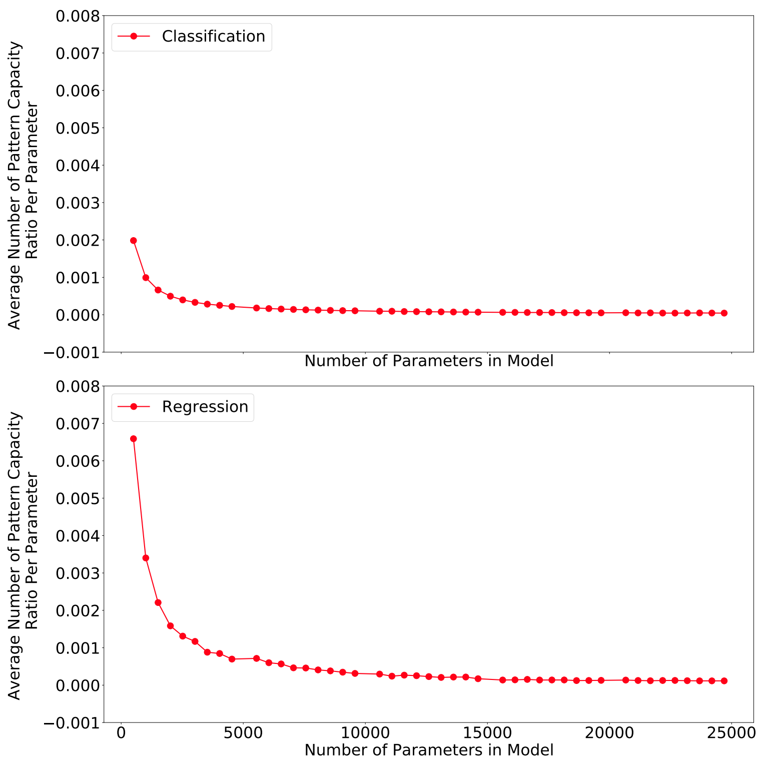

FIGURE 7.120: **Effect of increasing the number of parameters on the per parameter capacity of the network using RPROP training algorithm**

The average per parameter pattern capacity ratio using different training algorithms for each RNN, is presented in table 7.17. From table 7.17 initially, as in all other experiments the RNNs retained less information per parameter for the classification experiments than the regression experiments. Comparing the per parameter pattern capacity between all RNNs, it is found that in the regression case the bidirectional LSTM trained using the Adam training algorithm was able to achieve the highest ratio of 0.00051 followed by the bidirectional GRU also trained using Adam which achieved a per parameter pattern retention ratio of 0.00050. In the classification experiment the same configurations performed best with per parameter pattern retention ratios of 0.00012 and 0.00010 respectively. The Adam training algorithm allows for more granular weight updates in the current time step, which is probably why the RNNs trained are able to use their parameters more effectively when applied to the pattern retention problem statement.

		Training Algorithm			
RNN		Adam	Nesterov	RPROP	SGD
Bidirectional Elman	Classification	0.00011	0.00009	0.00009	0.00009
	Regression	0.00024	0.00011	0.00014	0.00011
Bidirectional GRU	Classification	0.00012	0.00010	0.00011	0.00010
	Regression	0.00050	0.00019	0.00039	0.00018
Bidirectional Jordan	Classification	0.00010	0.00010	0.00009	0.00010
	Regression	0.00022	0.00012	0.00015	0.00011
Bidirectional LSTM	Classification	0.00012	0.00010	0.00010	0.00010
	Regression	0.00051	0.00019	0.00042	0.00018
Elman	Classification	0.00012	0.00010	0.00010	0.00010
	Regression	0.00027	0.00011	0.00015	0.00012
GRU	Classification	0.00012	0.00010	0.00010	0.00010
	Regression	0.00047	0.00015	0.00033	0.00016
Jordan	Classification	0.00010	0.00009	0.00009	0.00009
	Regression	0.00020	0.00011	0.00014	0.00012
LSTM	Classification	0.00011	0.00010	0.00010	0.00010
	Regression	0.00049	0.00017	0.00041	0.00017

TABLE 7.17: Per parameter capacity between the number of patterns retained and the number of network parameters using different training algorithms for each RNN

7.3 Experiment 3: Frame of Retention

Experiment three combines experiment one, length of sequence retention, and experiment two, the number of patterns retained. Experiment three is designed to determine how many patterns of increasing sequence length can be retained as described in section 7.2. Here the retention ratio will be referred to as the frame capacity of retention per parameter. This experiment differs from the previous as in the current experiment there was only a regression experiment conducted. The reasoning behind using only the regression component of previous experiments, is that RNNs performed better in the regression experiments than the classification experiment for both the length of sequence retention and the number of patterns retained experiment.

This analysis follows the same format as 7.1 and 7.2, that is a holistic view over all RNNs for the frame capacity is investigated in section 7.3.1. The effect of using different activation functions on the frame retention capability of RNNs is evaluated in section 7.3.2. Thereafter the relationship between the per parameter pattern capacity for increasing number of hidden layers is investigated in section 7.3.3. Closing this section off, the influence of different training algorithms are inspected in section 7.3.4.

7.3.1 Average Per Parameter Capacity Retention for each RNN

This section investigates the frame capacity per parameter for each RNN overall.

Figures 7.121, 7.122, 7.123, 7.124, 7.125, 7.126, 7.127 and 7.128 presents the average frame retained by each RNN for increasing number of parameters in the networks. In the aforementioned figures, a frame of two patterns of time length two will be represented by a 2 on the y-axis. Considering the general trend observed in figures 7.121, 7.122, 7.123, 7.124, 7.125, 7.126, 7.127 and 7.128, it is clear that all RNNs struggled with frame retention. The struggle RNNs face with frame retention, may be a result of the way in which the experiment was structured, as RNNs are generally performant when applied to time-series retention. Regardless of the difficulties faced by the RNNs, the results from the frame retention experiments can still be compared in relative terms. Continuing the analysis, it is observed that bidirectional layers assisted the GRU, Jordan and LSTM

RNNs to increase frame retention. However, the same is not true for the Elman RNN. To further investigate the relationship between frame retention and model parameters for each RNN, inspection of table 7.18 is required.

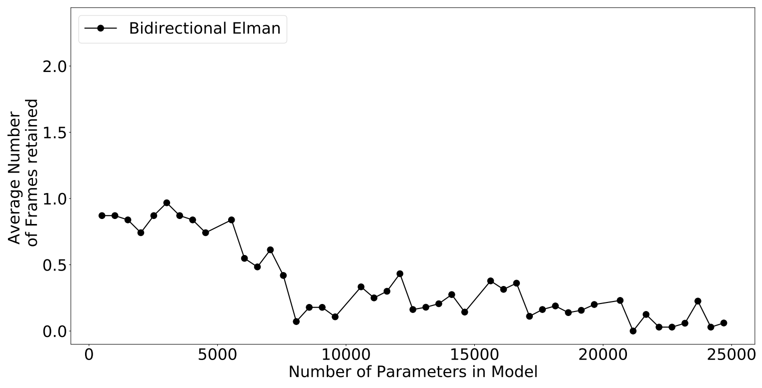

FIGURE 7.121: **Effect of the number of parameters on the ability to retain patterns for the bidirectional Elman RNN**

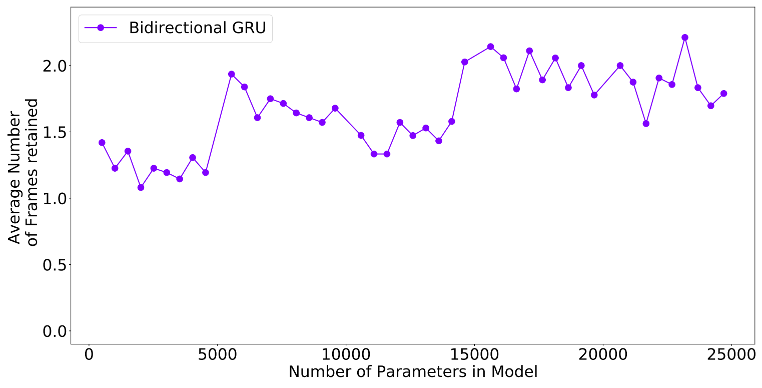

FIGURE 7.122: **Effect of the number of parameters on the ability to retain patterns for the bidirectional GRU RNN**

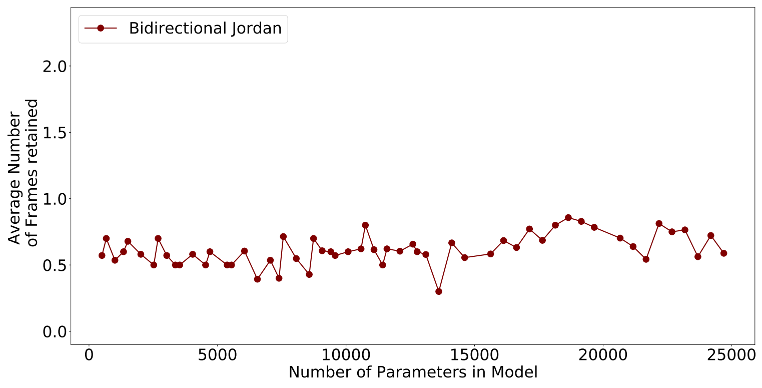

FIGURE 7.123: **Effect of the number of parameters on the ability to retain patterns for the bidirectional Jordan RNN**

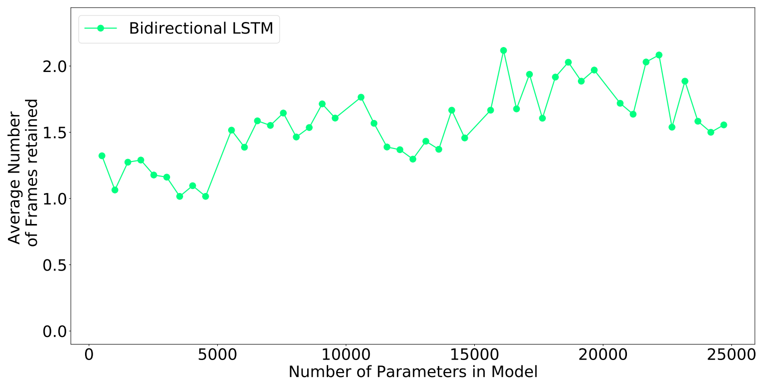

FIGURE 7.124: **Effect of the number of parameters on the ability to retain patterns for the bidirectional LSTM RNN**

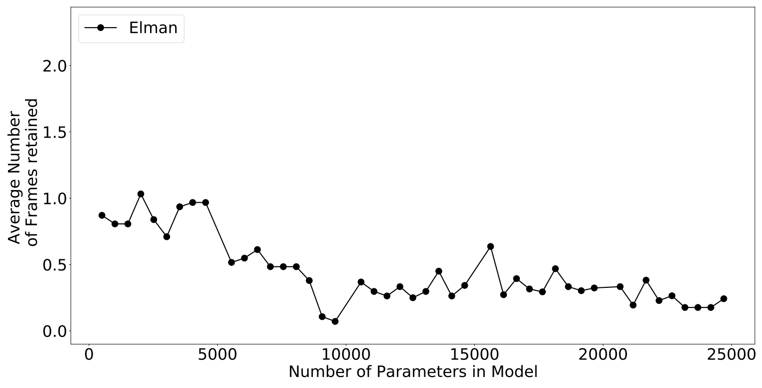

FIGURE 7.125: **Effect of the number of parameters on the ability to retain patterns for the Elman RNN**

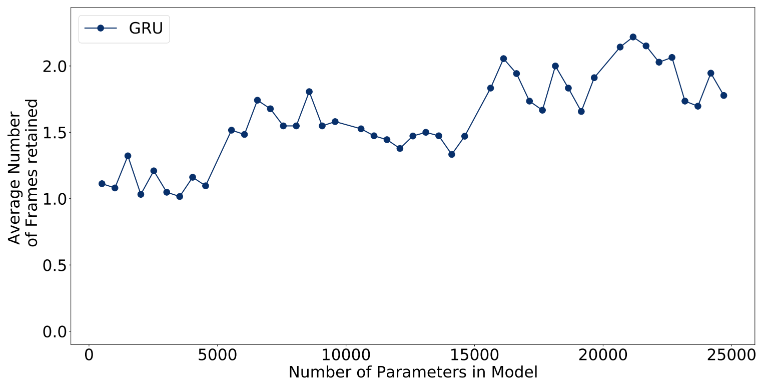

FIGURE 7.126: **Effect of the number of parameters on the ability to retain patterns for the GRU RNN**

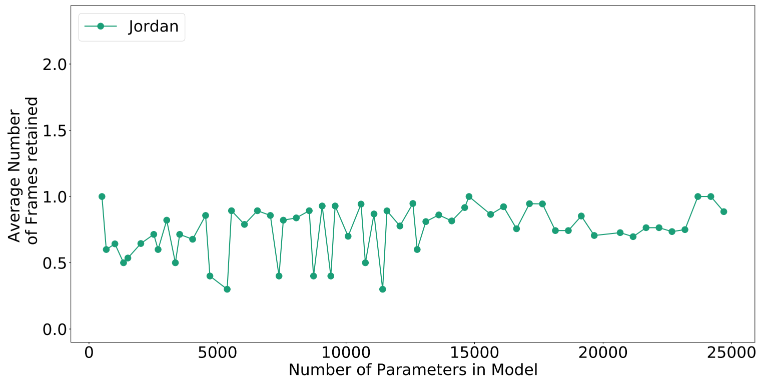

FIGURE 7.127: **Effect of the number of parameters on the ability to retain patterns for the Jordan RNN**

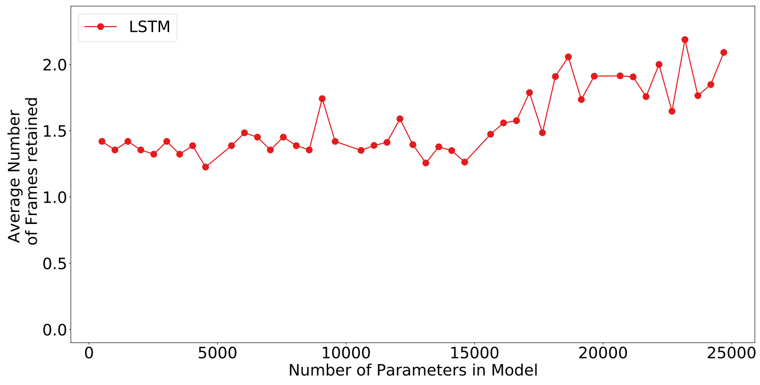

FIGURE 7.128: **Effect of the number of parameters on the ability to retain patterns for the LSTM RNN**

Table 7.18 presents the results of taking the Spearman Rank correlation between the number of parameters in the RNNs and the number of frames retained. In table 7.18 the relationships are more clear. As previously stated, the networks which had the worst correlation coefficients were the Elman and the bidirectional Elman RNNs with -0.26456 and -0.31323 respectively. From table 7.18 it is observed for the LSTM and Jordan RNNs that the correlation coefficients improved positively from 0.21015 and 0.09847 to 0.21015 and 0.11517 when using the bidirectional layers. Surprisingly, the correlation coefficient for the bidirectional GRU is less, at 0.21634, than that of the GRU, at 0.24909, indicating that the GRU benefits more from an increase in the number of parameters. Now that the relationship between the increase in the number of parameters and the frame retention capacity of RNNs have been explored, the per parameter frame capacity of the various networks will be inspected.

RNN	Spearman correlation
Bidirectional Elman	-0.31323
Bidirectional GRU	0.21634
Bidirectional Jordan	0.11517
Bidirectional LSTM	0.21015
Elman	-0.26456
GRU	0.24909
Jordan	0.09847
LSTM	0.20880

TABLE 7.18: Spearman correlation between the number of frames retained and the number of network parameters

Figures 7.129, 7.130, 7.131, 7.132, 7.133, 7.134, 7.135 and 7.136 convey the per parameter frame capacity as the number of parameters in the respective RNNs increase. As expected, and observed in the previous experiments, the per parameter capacity decreases as the number of parameters increase. This illustrates that the rate of decay in per parameter frame capacity is the highest with a few parameters, thereafter the per parameter capacity follows a logarithmic decay as the number of parameters increase.

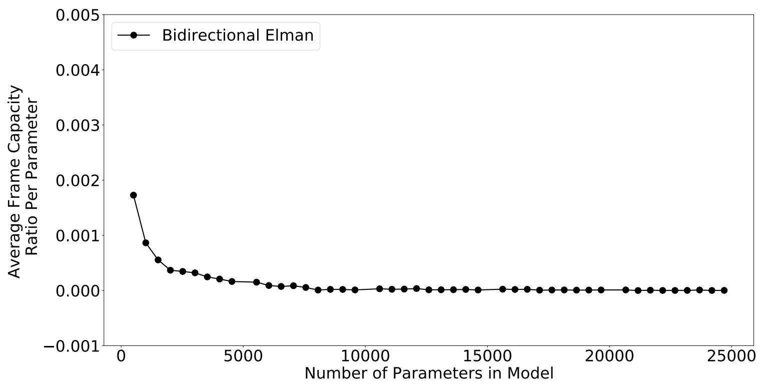

FIGURE 7.129: **Effect of increasing the number of parameters on the per parameter capacity of the Bidirectional Elman RNN**

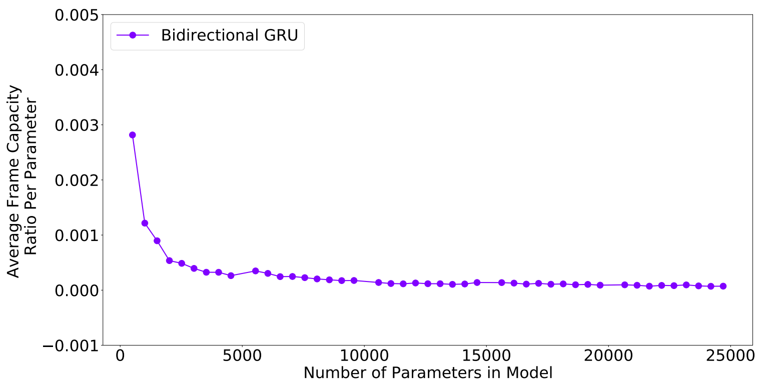

FIGURE 7.130: **Effect of increasing the number of parameters on the per parameter capacity of the Bidirectional GRU RNN**

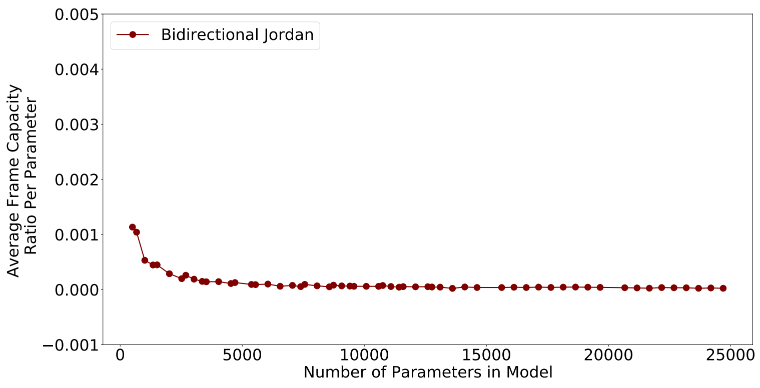

FIGURE 7.131: **Effect of increasing the number of parameters on the per parameter capacity of the bidirectional Jordan RNN**

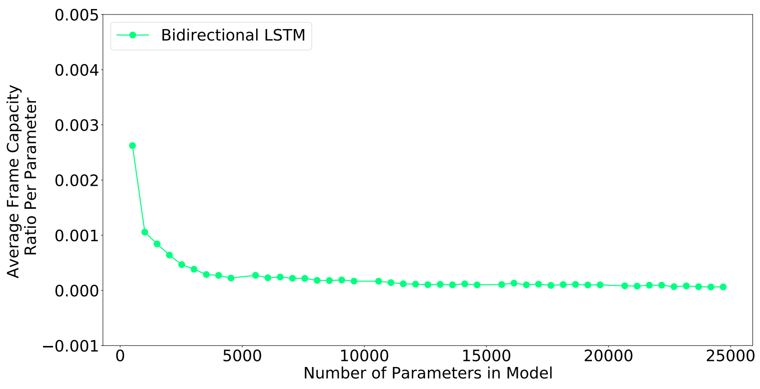

FIGURE 7.132: **Effect of increasing the number of parameters on the per parameter capacity of the Bidirectional LSTM RNN**

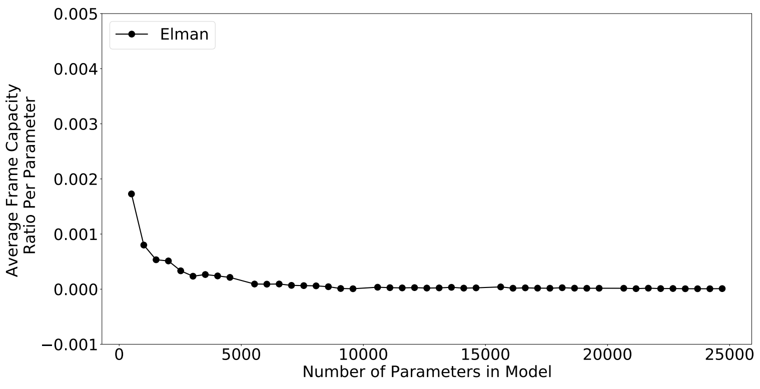

FIGURE 7.133: **Effect of increasing the number of parameters on the per parameter capacity of the Elman RNN**

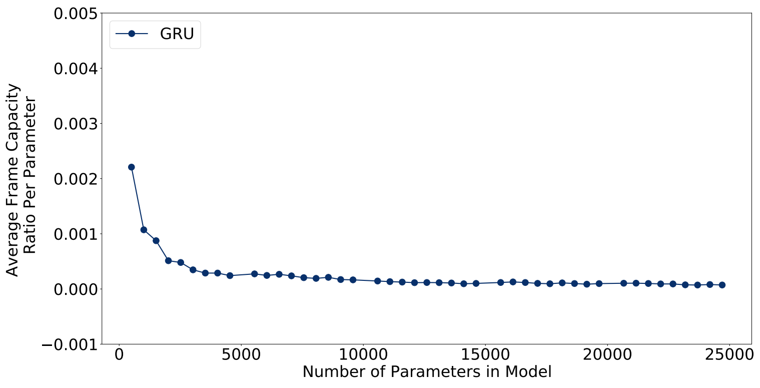

FIGURE 7.134: **Effect of increasing the number of parameters on the per parameter capacity of the GRU RNN**

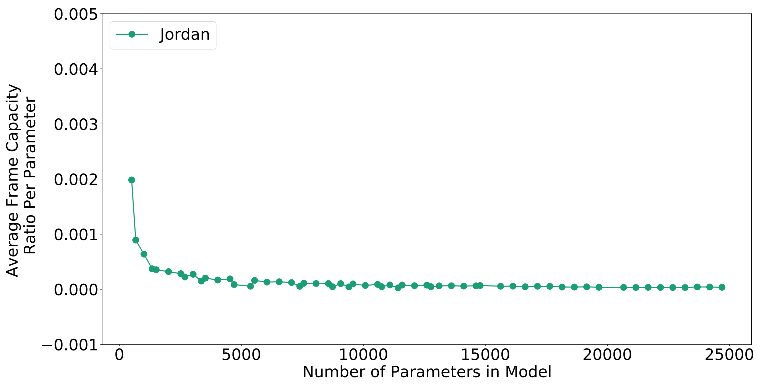

FIGURE 7.135: **Effect of increasing the number of parameters on the per parameter capacity of the Jordan RNN**

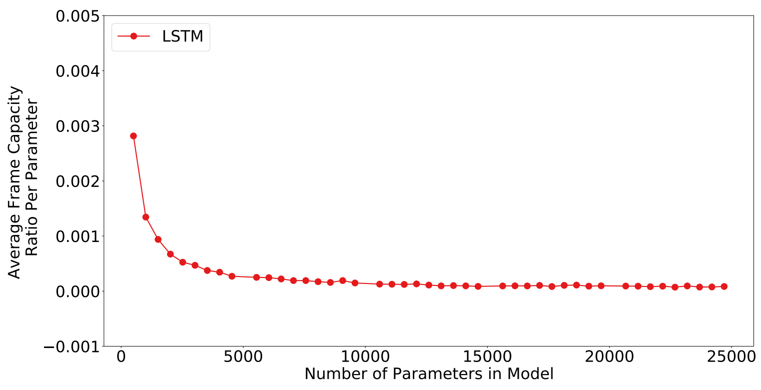

FIGURE 7.136: **Effect of increasing the number of parameters on the per parameter capacity of the LSTM RNN**

In table 7.19 the per parameter frame capacity for each RNN is presented. Here, the bidirectional GRU was able to achieve the highest per parameter frame capacity of 0.0001286 followed by the GRU with a ratio of 0.0001244. Upon further inspection it is also noted that only in the Elman RNN's case, did the bidirectional layers not result in any performance gain. The bidirectional layers not attributing to performance gain is surprising and may be attributed to the experimental structure as the underlying data set does not have a specific pattern which can be learned indicating that bidirectional layers are more applicable to problems in which additional abstraction is required.

RNN	Capacity Ratio
Bidirectional Elman	0.0000270
Bidirectional GRU	0.0001286
Bidirectional Jordan	0.0000491
Bidirectional LSTM	0.0001200
Elman	0.0000338
GRU	0.0001244
Jordan	0.0000632
LSTM	0.0001213

TABLE 7.19: Average number of frames retained per single network parameter

7.3.2 Average Per Parameter Capacity Retention for each Activation Function

Furthering the investigation, this section serves to investigate how activation functions affect the per parameter frame capacity of the different RNNs. The intention is to find the optimal activation function RNN pair to maximise the per parameter performance.

To determine the relationship between activation functions and frame capacity as the number of parameters grow, the results of the experiments conducted are presented in figures 7.137, 7.138, 7.139, 7.140, 7.141, 7.142, 7.143, 7.144, 7.145 and 7.146. The aforementioned figures report the average frame which was retained. From the listed figures it is seen that some activation functions lead to a larger frame retention capacity over all RNNs on average. The elu, linear, relu, selu and tanh functions handle increasing the number of parameters in the model better than the other activation functions. However, the variance in frame retention is much higher for the elu, linear, relu, selu and tanh activation functions. To further investigate the impact that these activation functions have on the frame retention rate as the number of model parameters increase, the Spearman rank correlation is considered in table 7.20.

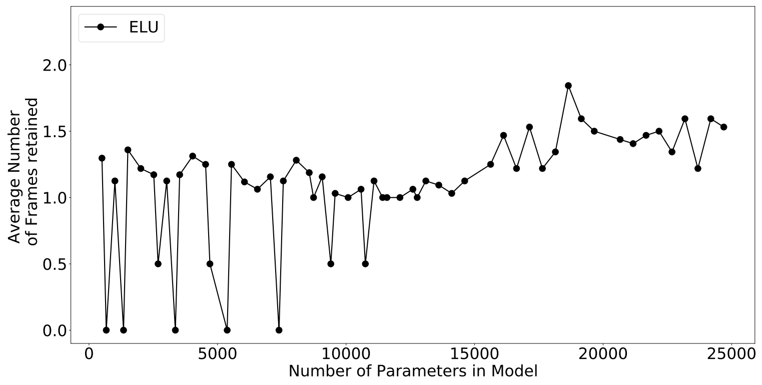

FIGURE 7.137: **Effect of using the Elu activation function on the frame retention over all RNNs for increasing parameters in the network architectures**

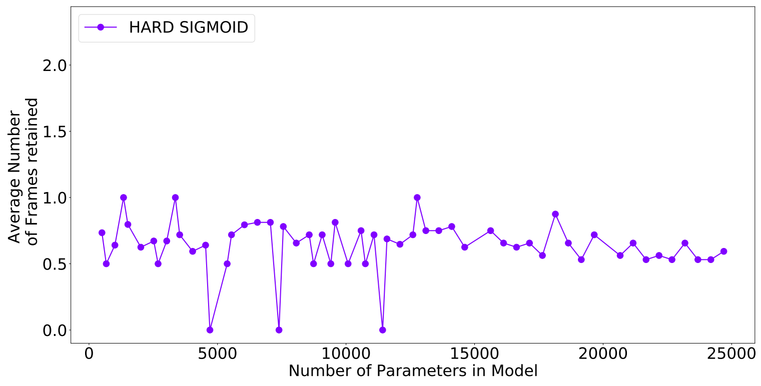

FIGURE 7.138: **Effect of using the Hard Sigmoid activation function on the frame retention over all RNNs for increasing parameters in the network architectures**

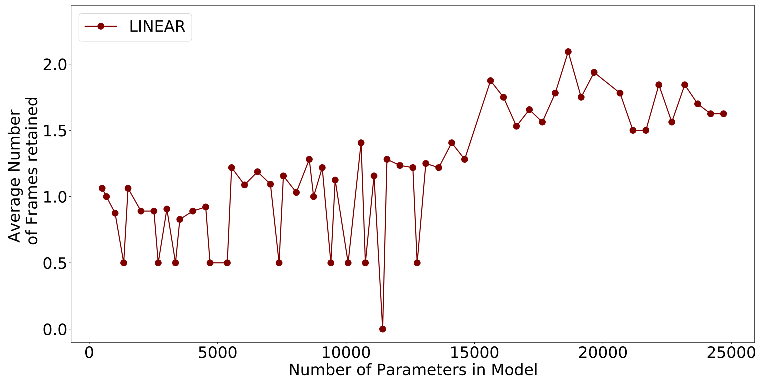

FIGURE 7.139: **Effect of using the Linear activation function on the frame retention over all RNNs for increasing parameters in the network architectures**

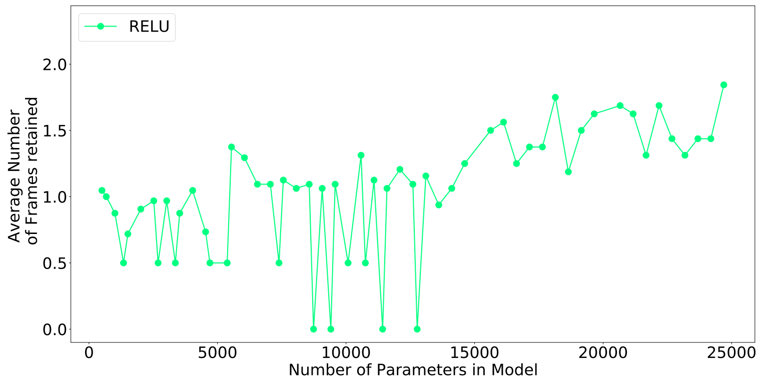

FIGURE 7.140: **Effect of using the Relu activation function on the frame retention over all RNNs for increasing parameters in the network architectures**

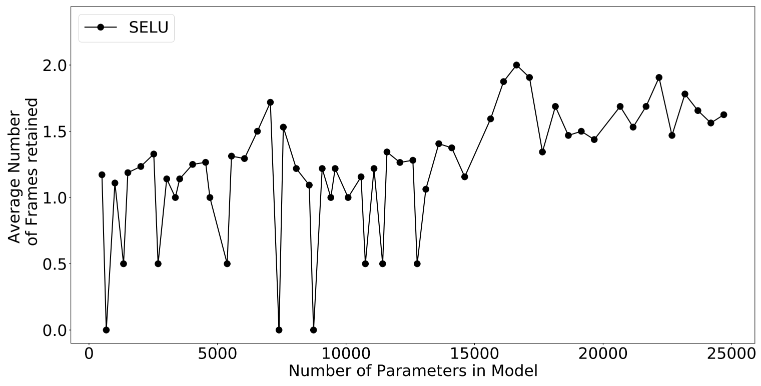

FIGURE 7.141: **Effect of using the Selu activation function on the frame retention over all RNNs for increasing parameters in the network architectures**

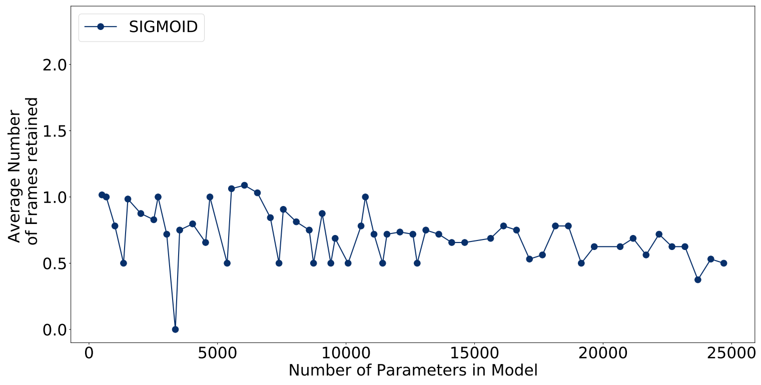

FIGURE 7.142: **Effect of using the Sigmoid activation function on the frame retention over all RNNs for increasing parameters in the network architectures**

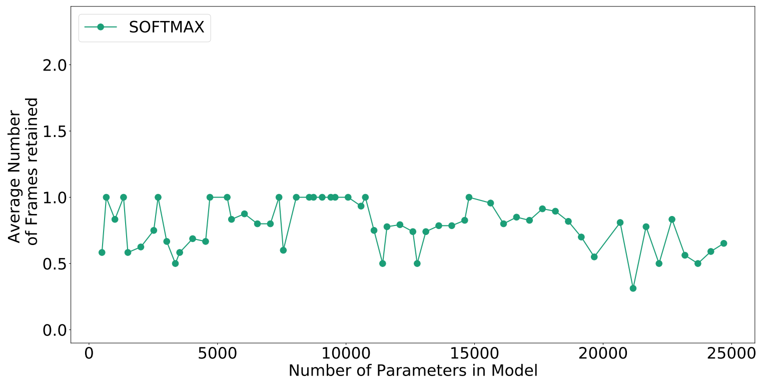

FIGURE 7.143: **Effect of using the Softmax activation function on the frame retention over all RNNs for increasing parameters in the network architectures**

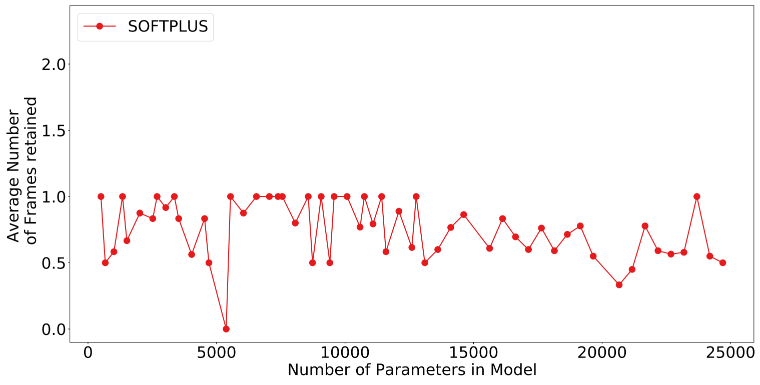

FIGURE 7.144: **Effect of using the Softplus activation function on the frame retention over all RNNs for increasing parameters in the network architectures**

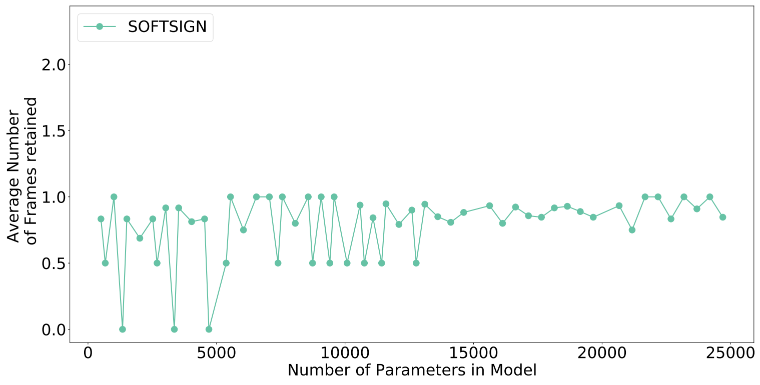

FIGURE 7.145: **Effect of using the Softsign activation function on the frame retention over all RNNs for increasing parameters in the network architectures**

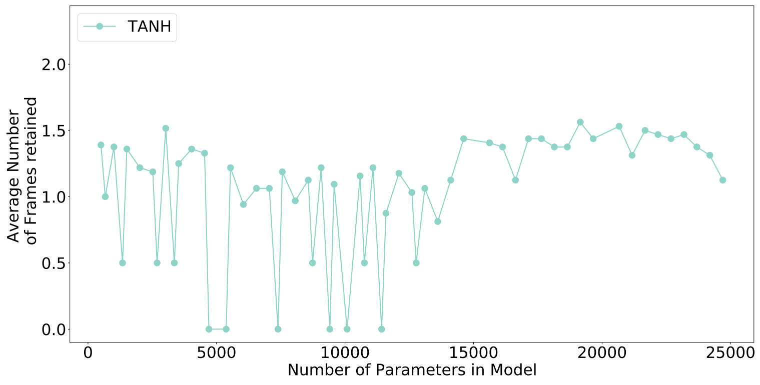

FIGURE 7.146: **Effect of using the Tanh activation function on the frame retention over all RNNs for increasing parameters in the network architectures**

Considering the Spearman rank correlation between frames retained and the number of parameters in the model, as presented in table 7.20, there is a clear negative correlation for the Elman RNN variants almost across the board. The elu, linear, relu, selu and tanh activation functions, which were found to be the most advantageous to use for frame retention, also show clear positive correlations especially with RNNs which performed well in section 7.3.1. The most positive relationship being obtained by using a GRU using the elu activation function, with a correlation coefficient of 0.6788, followed by the bidirectional GRU using relu, with a correlation coefficient of 0.6475. However, on average, the linear activation function had the highest correlation coefficient if measured across all RNNs with a ratio of 0.3476. To further the investigation, next an inspection on the per parameter frame capacity is performed.

To expose the per parameter frame capacity relationship for each activation function as the number of model parameters increase, figures 7.147, 7.148, 7.149, 7.150, 7.151, 7.152, 7.153, 7.154, 7.155 and 7.156 are presented. These figures illustrate the smaller the per parameter capacity becomes the more complex the problem the network tries to learn. On average the same behaviour is observed in the previously mentioned figures as in section 7.3.1; as the decay is logarithmic for all activation functions. To gauge the frame retention capacity per parameter, table 7.21 needs to be considered.

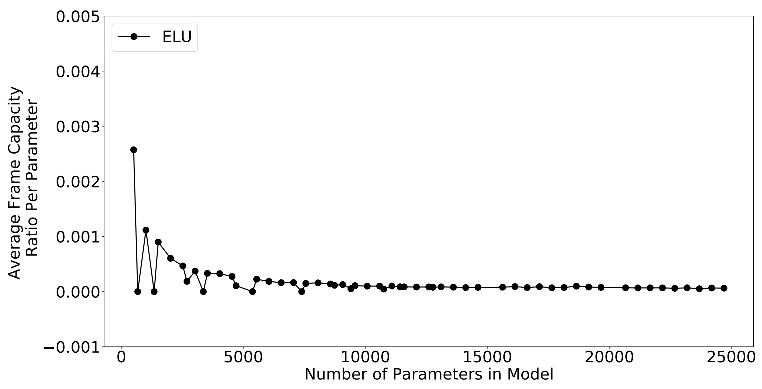

FIGURE 7.147: **Effect of increasing the number of parameters on the per parameter capacity of the network using elu activation function**

Activation Function	Bidirectional Elman	Bidirectional GRU	Bidirectional Jordan	Bidirectional LSTM	Elman	GRU	Jordan	LSTM
elu	-0.6690	0.2621	0.0376	0.3315	-0.6225	0.6788	0.3867	0.4818
hard sigmoid	-0.2649	-0.1420	0.0301	-0.1539	-0.0096	-0.0531	0.1679	-0.4239
linear	0.2827	0.5242	0.3196	0.4185	0.2584	0.5823	-0.0742	0.4694
relu	-0.3800	0.6475	0.2360	0.4439	-0.3780	0.6398	0.2716	0.4826
selu	-0.2923	0.3478	0.2492	0.3834	-0.1754	0.5218	-0.4411	0.4465
sigmoid	-0.3118	-0.2942	-0.1508	-0.3156	-0.3379	-0.3223	0.3140	-0.3281
softmax	0.2465	0.4910	-0.1387	0.5330	-0.4764	-0.4538	0.0000	0.0000
softplus	-0.8015	0.4644	0.0259	0.4895	-0.0329	0.5062	0.3335	0.0000
softsign	0.0000	0.5015	0.3258	0.4398	0.0000	0.4392	0.1846	0.0000
tanh	-0.6150	0.4346	0.1124	0.5713	-0.6232	0.1788	0.3514	0.3977

TABLE 7.20: Spearman correlation between the number of frames retained and the number of network parameters per activation function for each RNN

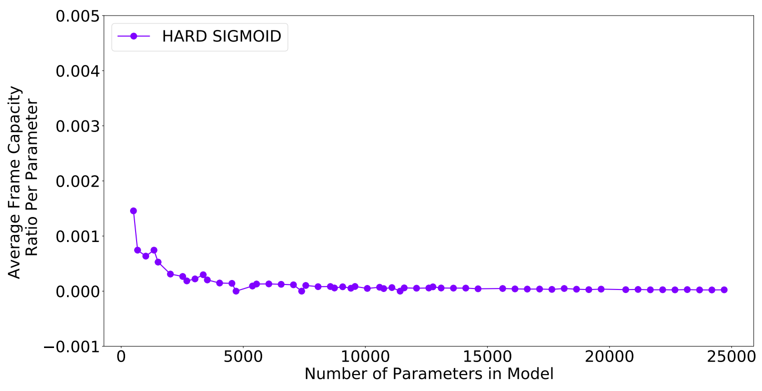

FIGURE 7.148: **Effect of increasing the number of parameters on the per parameter capacity of the network using hard sigmoid activation function**

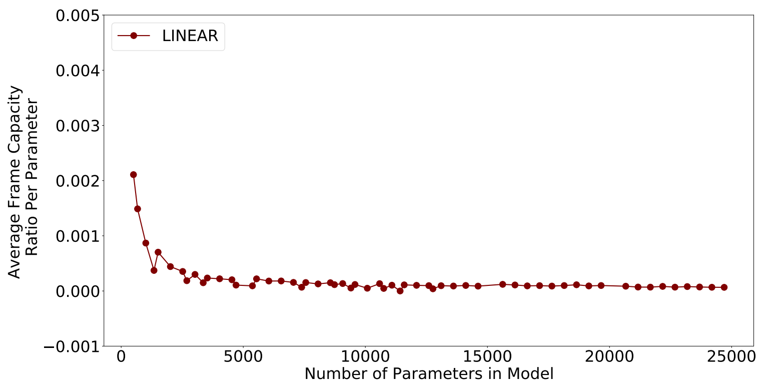

FIGURE 7.149: **Effect of increasing the number of parameters on the per parameter capacity of the network using linear activation function**

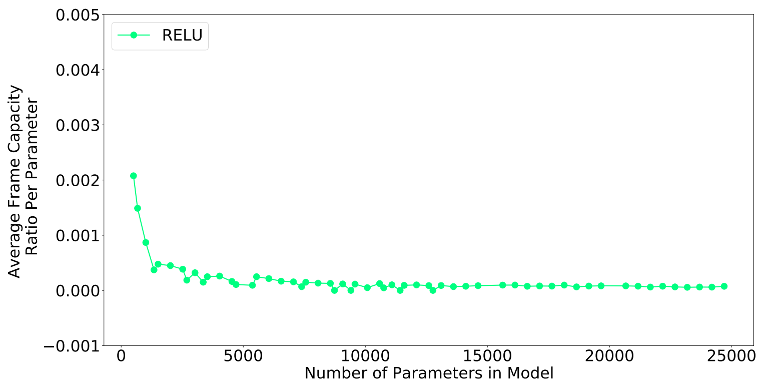

FIGURE 7.150: **Effect of increasing the number of parameters on the per parameter capacity of the network using relu activation function**

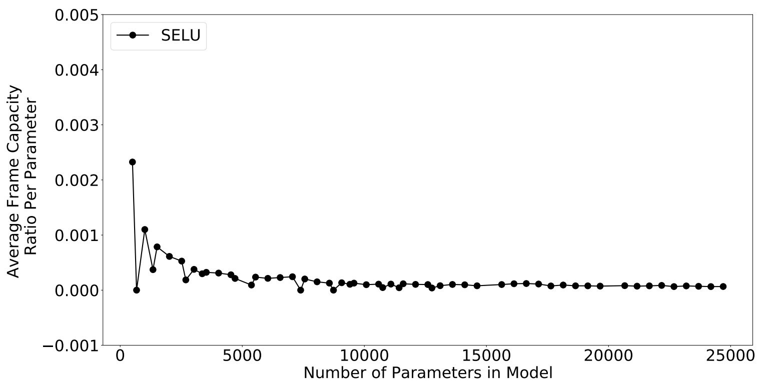

FIGURE 7.151: **Effect of increasing the number of parameters on the per parameter capacity of the network using selu activation function**

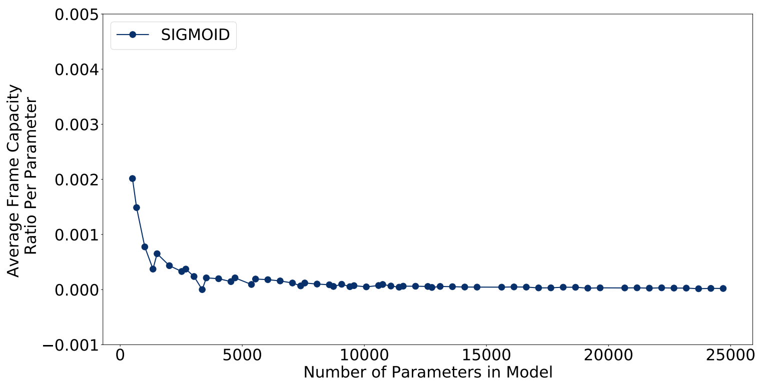

FIGURE 7.152: **Effect of increasing the number of parameters on the per parameter capacity of the network using sigmoid activation function**

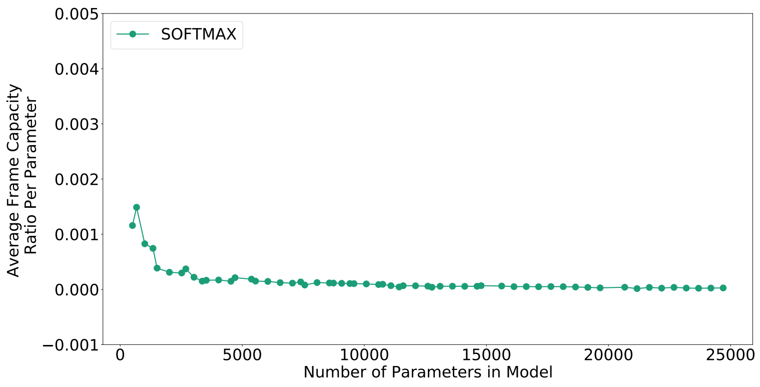

FIGURE 7.153: **Effect of increasing the number of parameters on the per parameter capacity of the network using softmax activation function**

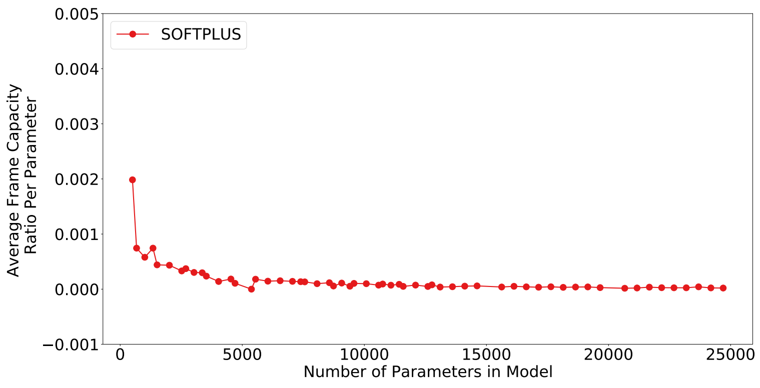

FIGURE 7.154: **Effect of increasing the number of parameters on the per parameter capacity of the network using softplus activation function**

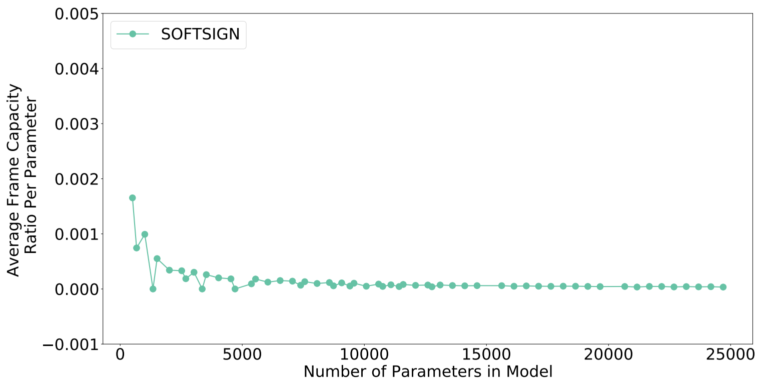

FIGURE 7.155: **Effect of increasing the number of parameters on the per parameter capacity of the network using softsign activation function**

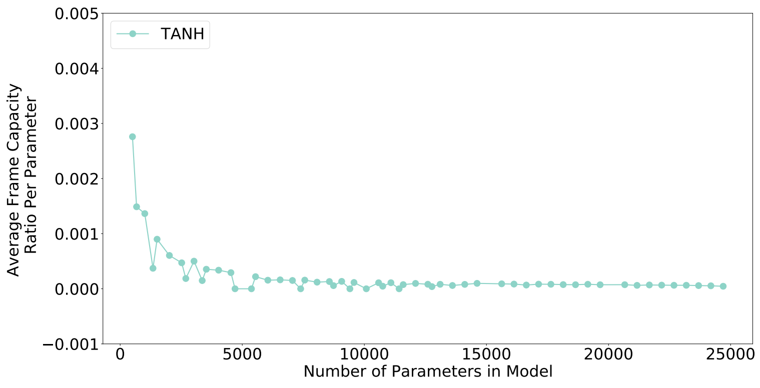

FIGURE 7.156: **Effect of increasing the number of parameters on the per parameter capacity of the network using tanh activation function**

Table 7.21 exposes the per parameter frame capacity for each activation function for each RNN. Considering average performance over all RNNs, it is clear that the selu activation function had the highest per parameter frame capacity of 0.0001132 overall. Upon further inspection, the highest per parameter frame capacity achieved was by the LSTM using the selu activation function of 0.0001132, followed closely by the bidirectional GRU also using relu with a per parameter frame retention ratio of 0.0001889. The selu activation function performing better than other activation functions is not surprising as this activation function has performed well in the previous experiments.

7.3.3 Relationship between the Number of Frames and the Number Of Layers

Furthering the investigation of the different components affecting the frame capacity of RNNs, this section investigates how increasing the number of hidden layers affects the per parameter frame capacity.

Figures 7.157, 7.158, 7.159, 7.160, 7.161, 7.162, 7.163 and 7.164 present the average frame retained per hidden layer added. Inspection of figures 7.157, 7.158, 7.159, 7.160, 7.161, 7.162, 7.163 and 7.164 reveals that increasing the number of layers benefited some RNNs, but reduced retention in others. As can be observed, the bidirectional Elman, bidirectional Jordan RNN and Elman RNNs retained less frames on average the more hidden layers were added. Not only did the frame retention problem prove difficult for these subsets of networks, but by increasing the number of layers, the networks essentially reduced in cognitive ability. In the case of all other RNNs, increasing the number of layers seemingly did increase the number of frames retained, however further analysis is required.

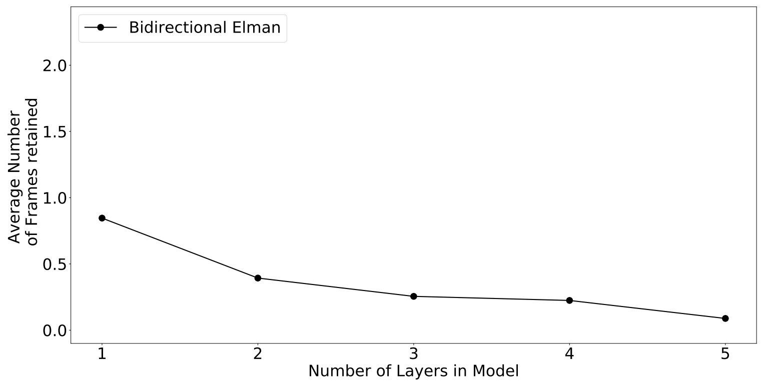

FIGURE 7.157: **Effect of increasing number of layers the on the ability to retain patterns for the Bidirectional Elman RNN**

Activation Function	Bidirectional Elman	Bidirectional GRU	Bidirectional Jordan	Bidirectional LSTM	Elman	GRU	Jordan	LSTM	Average
elu	0.0000278	0.0001446	0.0000467	0.0001581	0.0000282	0.0001748	0.0000686	0.0001583	0.0001009
hard sigmoid	0.0000044	0.0000736	0.0000434	0.0000730	0.0000093	0.0000787	0.0000771	0.0000728	0.0000540
linear	0.0000608	0.0001736	0.0000632	0.0001548	0.0000635	0.0001574	0.0000408	0.0001534	0.0001084
relu	0.0000229	0.0001889	0.0000480	0.0001093	0.0000243	0.0001856	0.0000741	0.0001314	0.0000981
selu	0.0000410	0.0001545	0.0000438	0.0001775	0.0000551	0.0001810	0.0000543	0.0001984	0.0001132
sigmoid	0.0000062	0.0000928	0.0000493	0.0000836	0.0000062	0.0000789	0.0000741	0.0000798	0.0000589
softmax	0.0000411	0.0000643	0.0000498	0.0000633	0.0000132	0.0000382	0.0000660	0.0000686	0.0000506
softplus	-0.0000411	0.0000645	0.0000478	0.0000626	0.0000527	0.0000658	0.0000490	0.0000701	0.0000464
softsign	0.0000746	0.0000679	0.0000385	0.0000661	0.0000731	0.0000749	0.0000401	0.0000737	0.0000636
tanh	0.0000401	0.0001675	0.0000556	0.0001722	0.0000353	0.0001248	0.0000707	0.0001349	0.0001001

Table 7.21: Per parameter capacity between number of frames retained and the number of network parameters per activation function for each RNN

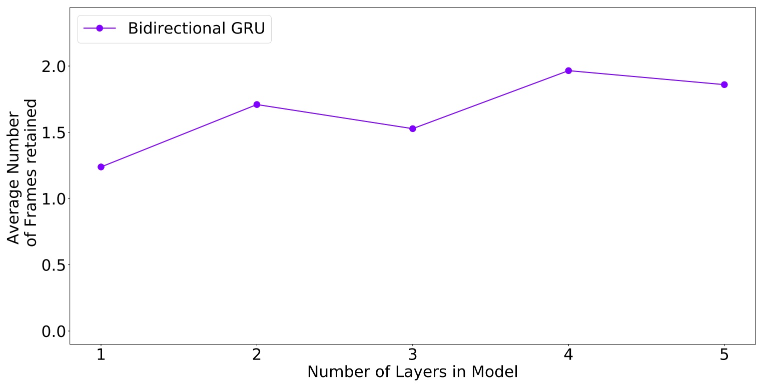

FIGURE 7.158: **Effect of increasing number of layers the on the ability to retain patterns for the Bidirectional GRU RNN**

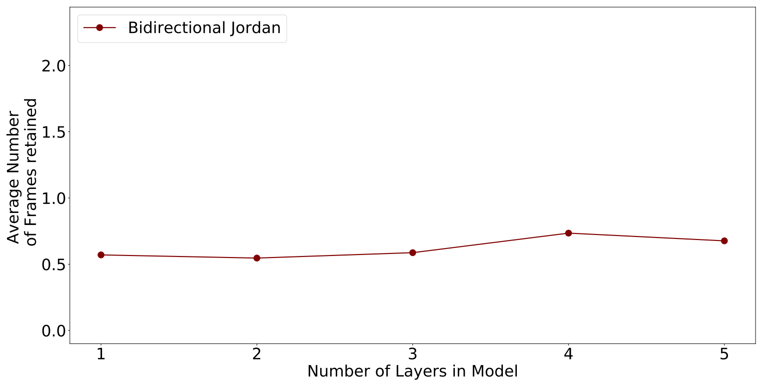

FIGURE 7.159: **Effect of increasing number of layers the on the ability to retain patterns for the Bidirectional Jordan RNN**

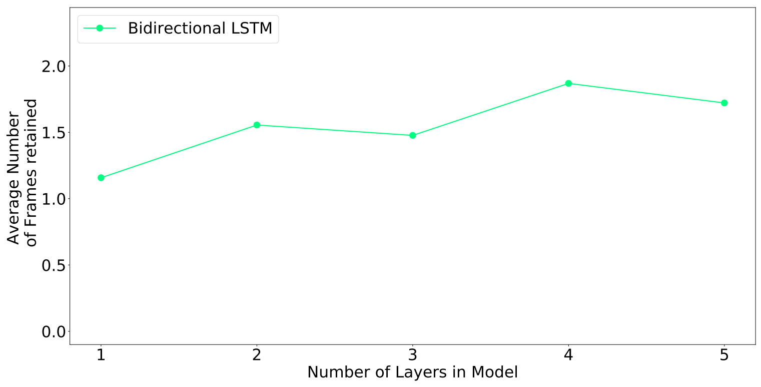

FIGURE 7.160: **Effect of increasing number of layers the on the ability to retain patterns for the Bidirectional LSTM RNN**

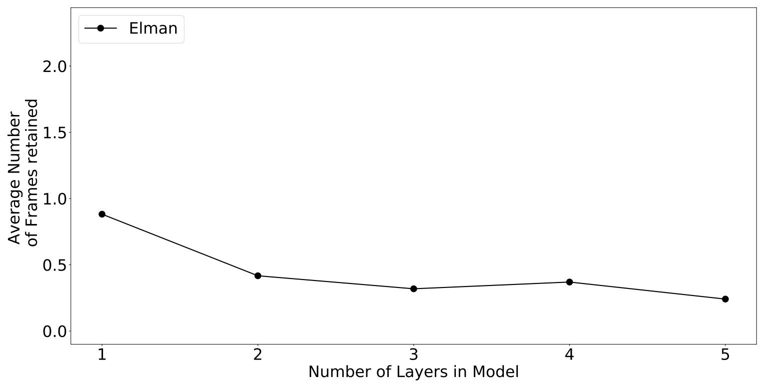

FIGURE 7.161: **Effect of increasing number of layers the on the ability to retain patterns for the Elman RNN**

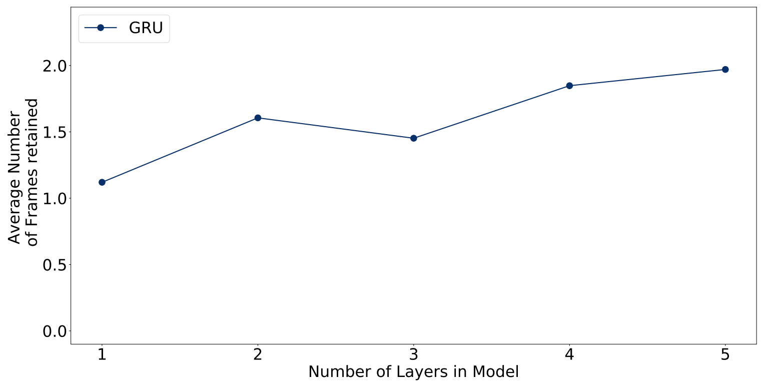

FIGURE 7.162: **Effect of increasing number of layers the on the ability to retain patterns for the GRU RNN**

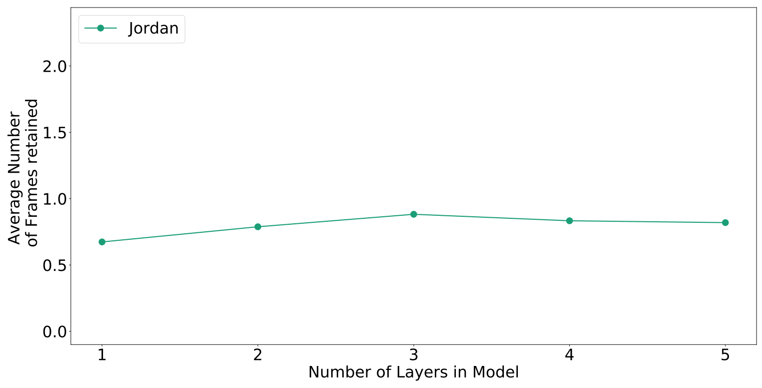

FIGURE 7.163: **Effect of increasing number of layers the on the ability to retain patterns for the Jordan RNN**

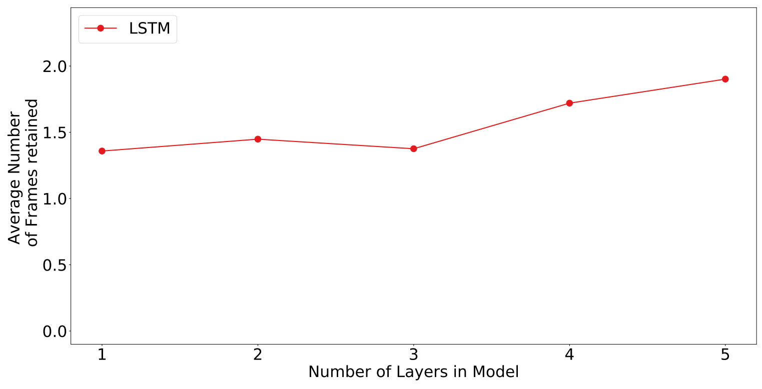

FIGURE 7.164: **Effect of increasing number of layers the on the ability to retain patterns for the LSTM RNN**

Table 7.22 represents the Spearman Rank correlation between the number of layers and the frame capacity per parameter for each RNN is investigated. From table 7.22 it can be observed that as the number of layers increase so does the per parameter frame capacity decrease on average. However, in some cases increasing the number of layers increases the per parameter frame capacity for the RNN. When referring to the bidirectional GRU at layer 3 the correlation coefficient increases to 0.1729 from -0.1031 as in layer 2. Other RNNs exhibit the same behaviour such as the bidirectional Jordan as well as the GRU to name a few. Since the RNNs perform better in some configurations than others, it may indicate that there exists some optimal configuration for each RNN, which is left for future work. The RNN which benefited most from increasing layer depth is the LSTM as the correlation coefficient of the per parameter frame capacity remains positive on average at 0.02294 indicating that the LSTM is more suited to be stacked in deeper layer configurations for the frame retention problem statement.

The per parameter frame capacity as the number of hidden layers increase for each RNN is shown in figures 7.165, 7.166, 7.167, 7.168, 7.169, 7.170, 7.171 and 7.172. From these figures, it is clear that when increasing the number of hidden layers, the amount of knowledge which can be retained per parameter is sacrificed. The drop in frame capacity is most prevalent between increasing from one hidden layer to two. This indicates that wider RNNs with fewer hidden layers would be able to retain a higher amount of frames than their deeper counter parts on average.

	Number of Layers					
RNNs	1	2	3	4	5	Average
Bidirectional Elman	-0.0147	-0.2979	-0.0780	-0.1057	-0.0286	-0.10498
Bidirectional GRU	-0.0512	-0.1031	0.1729	-0.0659	-0.0260	-0.01466
Bidirectional Jordan	-0.0296	0.0298	-0.0771	0.1670	-0.0162	0.01478
Bidirectional LSTM	-0.1043	0.0602	-0.0424	0.0519	-0.0952	-0.02596
Elman	0.0363	-0.1921	0.0105	-0.0992	-0.0756	-0.06402
GRU	-0.0273	0.0208	-0.0268	-0.0333	-0.1127	-0.03586
Jordan	-0.1179	-0.2735	-0.0509	-0.1238	0.2297	-0.06728
LSTM	-0.0497	0.0388	-0.0490	0.1480	0.0266	0.02294

TABLE 7.22: Spearman correlation between the number of frames retained and the number of network parameters for increasing NN depth for each RNN

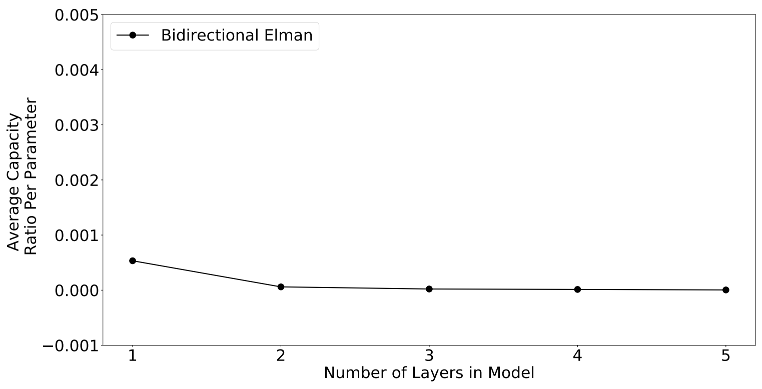

FIGURE 7.165: **Effect of increasing the number of parameters on the per parameter capacity of the network when increasing the number of layers for the bidirectional Elman RNN**

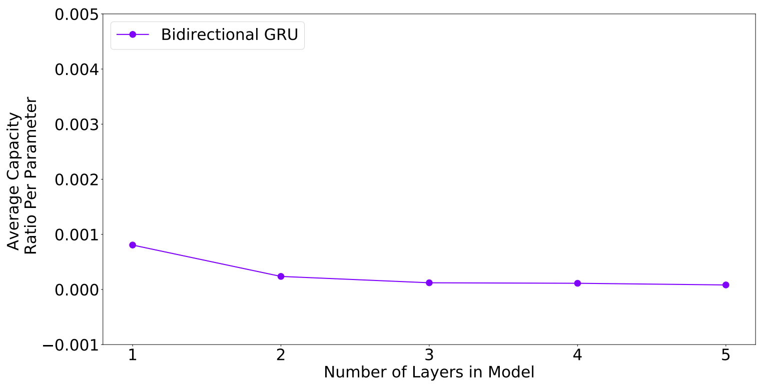

FIGURE 7.166: **Effect of increasing the number of parameters on the per parameter capacity of the network when increasing the number of layers for the Bidirectional GRU RNN**

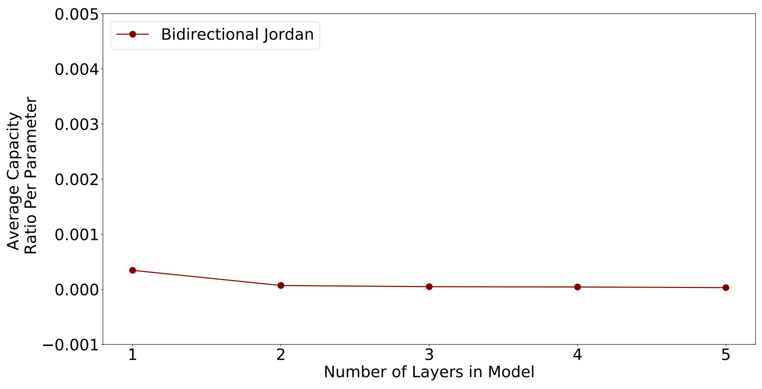

FIGURE 7.167: **Effect of increasing the number of parameters on the per parameter capacity of the network when increasing the number of layers for the Bidirectional Jordan RNN**

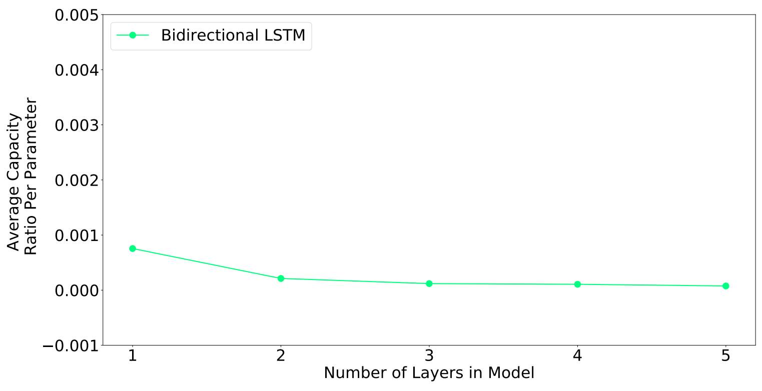

FIGURE 7.168: **Effect of increasing the number of parameters on the per parameter capacity of the network when increasing the number of layers for the Bidirectional LSTM RNN**

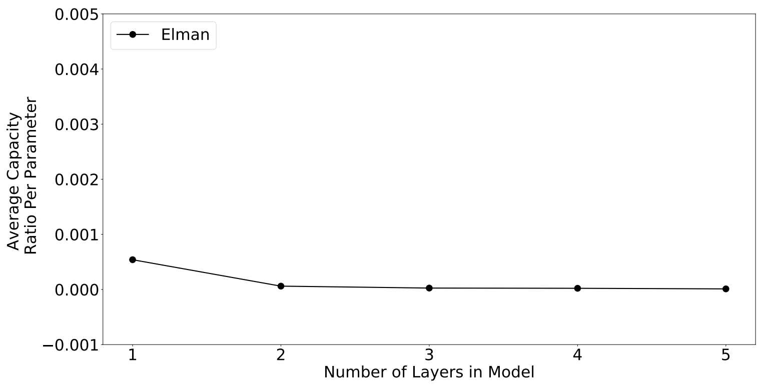

FIGURE 7.169: **Effect of increasing the number of parameters on the per parameter capacity of the network when increasing the number of layers for the Elman RNN**

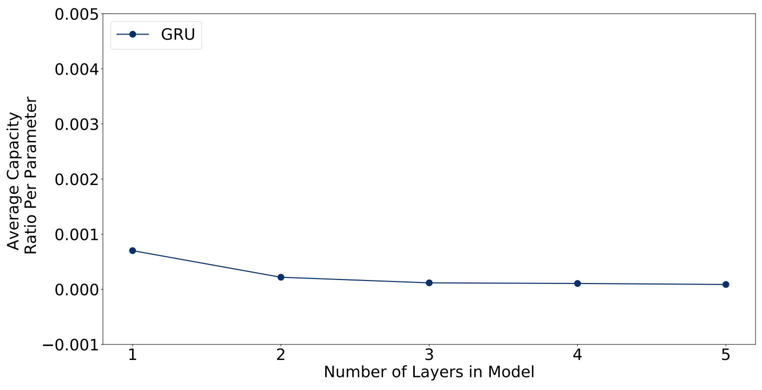

FIGURE 7.170: **Effect of increasing the number of parameters on the per parameter capacity of the network when increasing the number of layers for the GRU RNN**

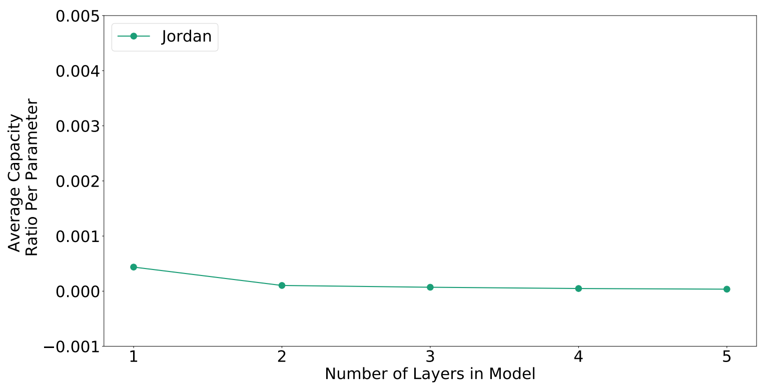

FIGURE 7.171: **Effect of increasing the number of parameters on the per parameter capacity of the network when increasing the number of layers for the Jordan RNN**

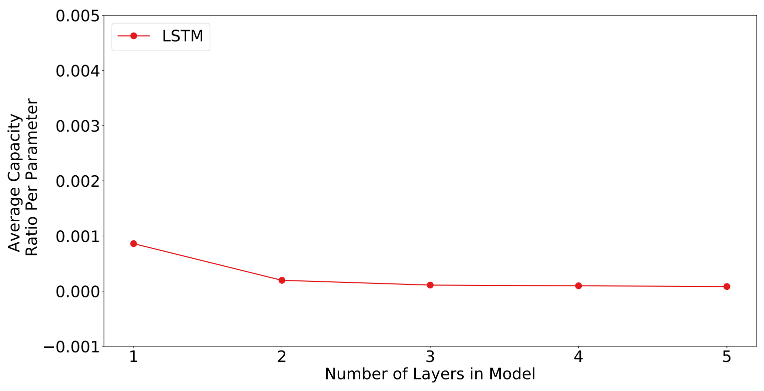

FIGURE 7.172: **Effect of increasing the number of parameters on the per parameter capacity of the network when increasing the number of layers for the LSTM RNN**

Table 7.23 presents the per frame capacity ratio for each RNN at increasing numbers of hidden layers. Overall the bidirectional GRU achieved the highest per parameter frame capacity of 0.0002058, followed by the LSTM of 0.0002036. If the number of hidden layers are examined separately and the performance of each RNN using that number of hidden layers, it can be observed that the LSTM had the highest capacity when one hidden layer was used with a per parameter frame retention ratio of 0.0005365. Thereafter the bidirectional GRU reached the best per parameter frame capacity of 0.0002267 when using two hidden layers, using three hidden layers with a ratio of 0.0001211 and four with a ratio of 0.0001121, among all RNNs using the same number of hidden layers. The GRU achieved the highest per parameter frame capacity of 0.000867 using five hidden layers when compared to the other RNNs. Considering these results, it is clear that wider RNNs would have a higher frame capacity for this experiment and that the bidirectional GRU is able to utilise an increasing number of hidden layers more effectively than other RNNs. Deeper RNNs lose more information as the layers increase in depth as compression of information is applied due to the loss in gradient information during back-propagation.

7.3.4 Relationship between the Number of Frames and Different Training Algorithms

This section explores the effects of using different training algorithms on the per parameter frame capacity.

Figures 7.173, 7.174, 7.175 and 7.176 present the average number of frames retained over all RNNs trained using the presented training algorithms. The number of frames retained has a higher variance for the Adam training algorithm as illustrated in figure 7.173, nevertheless the highest average number of frames was achieved using this method. The other training algorithms seem to be more consistent in their results and an increase in the average number of frames retained can be observed as the number of parameters in the network increases.

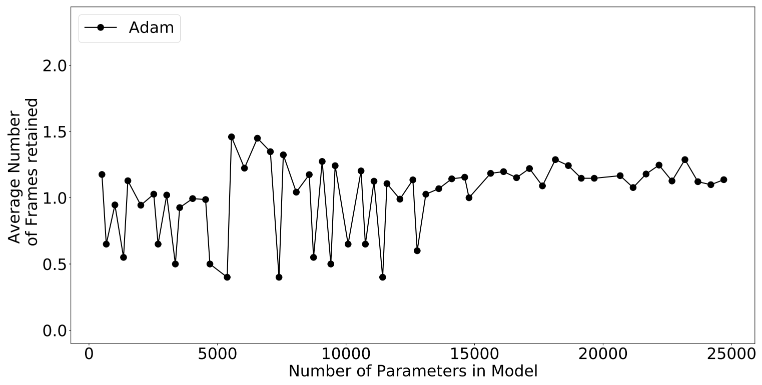

FIGURE 7.173: **Effect of using Adam training algorithm on the ability to retain patterns**

	Number of Layers					
RNNs	1	2	3	4	5	Average
Bidirectional Elman	0.0003349	0.0000524	0.0000203	0.0000127	0.0000039	0.0000848
Bidirectional GRU	0.0004873	0.0002267	0.0001211	0.0001112	0.0000818	0.0002056
Bidirectional Jordan	0.0002083	0.0000663	0.0000465	0.0000417	0.0000298	0.0000785
Bidirectional LSTM	0.0004551	0.0002063	0.0001170	0.0001057	0.0000757	0.0001920
Elman	0.0003495	0.0000555	0.0000253	0.0000210	0.0000106	0.0000924
GRU	0.0004429	0.0002121	0.0001153	0.0001048	0.0000867	0.0001924
Jordan	0.0002465	0.0000957	0.0000699	0.0000474	0.0000361	0.0000991
LSTM	0.0005365	0.0001912	0.0001088	0.0000975	0.0000838	0.0002036

TABLE 7.23: Per parameter capacity for each RNN per activation function at various layer depths

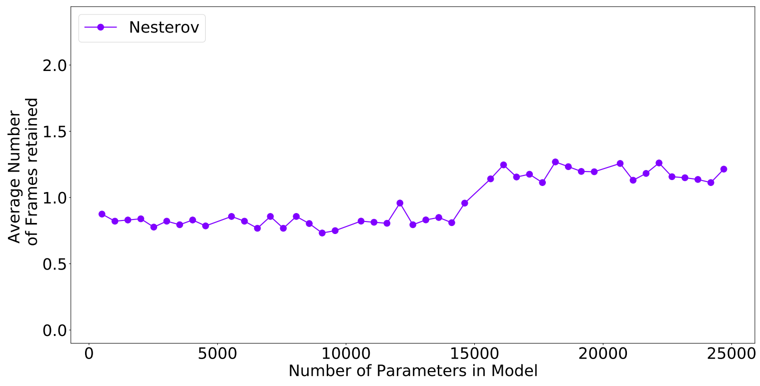

FIGURE 7.174: **Effect of using Nesterov training algorithm on the ability to retain patterns**

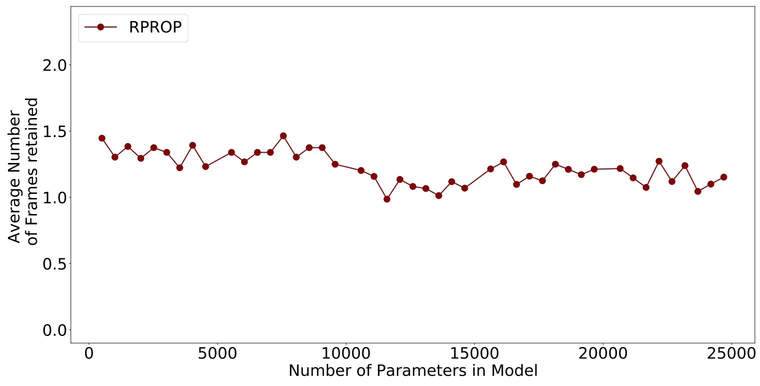

FIGURE 7.175: **Effect of using RPROP training algorithm on the ability to retain patterns**

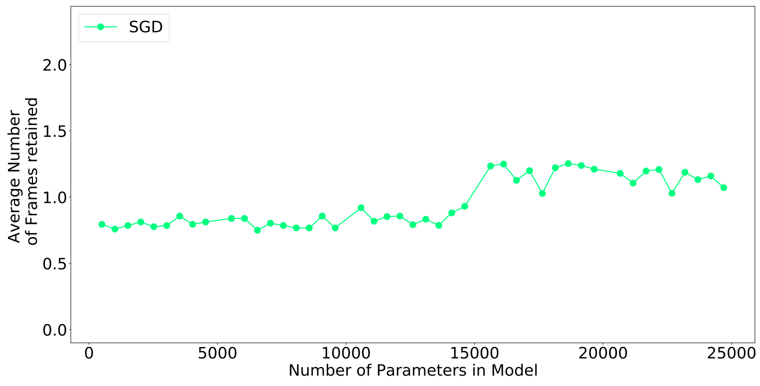

FIGURE 7.176: **Effect of using SGD training algorithm on the ability to retain patterns**

The Spearman Rank correlation between the number of frames retained and the number of parameters in the RNNs using different training algorithms, is presented in table 7.24. Upon inspection the most positive correlation coefficient of 0.4437 was achieved by the GRU trained using the Nesterov gradient descent training algorithm. Indicating that for the Nesterov training algorithm the GRU is the most responsive. The least responsive RNN for the Nesterov training algorithm is the Jordan RNN as it has the most negative correlation coefficient at -0.2613. The most negative correlation of -0.4338 was achieved when applying the Adam training algorithm to train the Elman bidirectional RNN. The Adam training algorithm achieved the most responsiveness from the Jordan RNN, with a correlation coefficient of 0.2597, followed by the LSTM, with a correlation coefficient of 0.2377. The RPROP achieved the most positive influence on the Jordan RNN, with a correlation coefficient of 0.1985, and the least on the bidirectional Elman, with a correlation coefficient of -0.3748. Lastly, the SGD training algorithm applied to the GRU had the highest correlation coefficient for the number of frames retained at 0.4369 and the most negative correlation coefficient of -0.2280, when applied to training the bidirectional Elman RNN. At a first glance, it would seem as if the Nesterov and SGD training algorithms would be able to achieve the highest frame capacity per parameter. However these numbers are affected by the low variance in the number of frames retained using these techniques. To determine which training algorithm lead to the highest per parameter frame capacity, further analysis of figures 7.177, 7.178, 7.179, 7.180 and table 7.25 is essential.

	Training Algorithms			
RNNs	Adam	Nesterov	RPROP	SGD
Bidirectional Elman	-0.4338	-0.1878	-0.3748	-0.2280
Bidirectional GRU	0.1194	0.4190	-0.0332	0.4453
Bidirectional Jordan	0.0692	0.1305	0.1085	0.1726
Bidirectional LSTM	0.1784	0.4380	-0.0767	0.4268
Elman	-0.3952	-0.1794	-0.3170	-0.1411
GRU	0.2377	0.4437	-0.0381	0.4369
Jordan	0.2597	-0.2613	0.1985	-0.2198
LSTM	0.2590	0.3948	-0.0932	0.3816

Table 7.24: Spearman correlation between Number of patterns retained and the number of network parameters using different training algorithms for each RNN

The results of measuring the per parameter frame capacity for increasing the number of parameters over the separate RNNs investigated is presented in figures 7.177, 7.178, 7.179 and 7.180. The decay in per parameter frame capacity follows the overall behaviour of the RNNs as described previously in section 7.3.1. Considering the slope of the Adam training algorithm, the variance seems to work in its favour. The other training algorithms have a smooth logarithmic decay, whereas the Adam training algorithm spikes up and down, which seemingly gives the Adam training algorithm a better chance at jumping out of local minima.

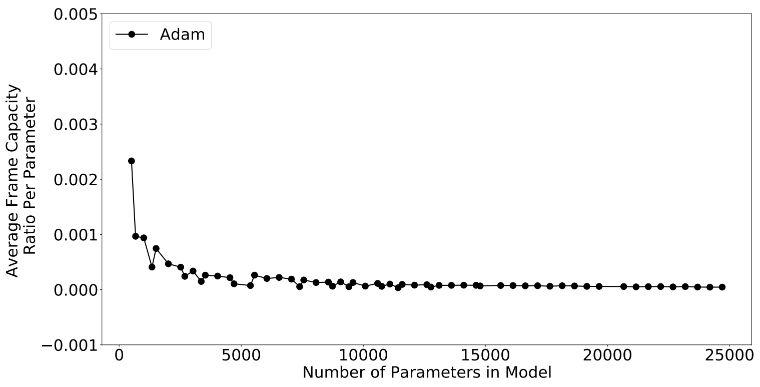

FIGURE 7.177: **Effect of increasing the number of parameters on the per parameter capacity of the network using Adam training algorithm**

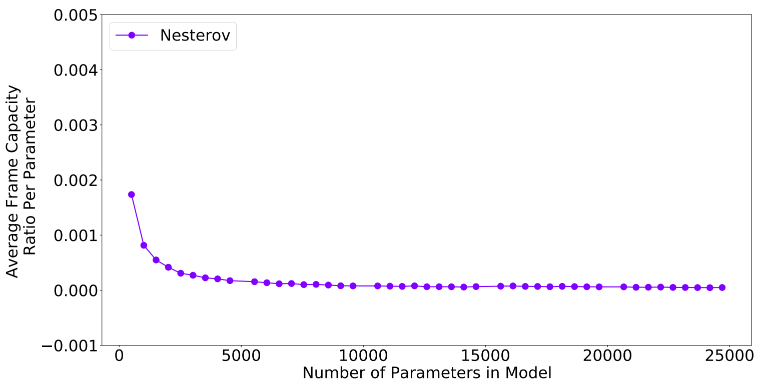

FIGURE 7.178: **Effect of increasing the number of parameters on the per parameter capacity of the network using Nesterov training algorithm**

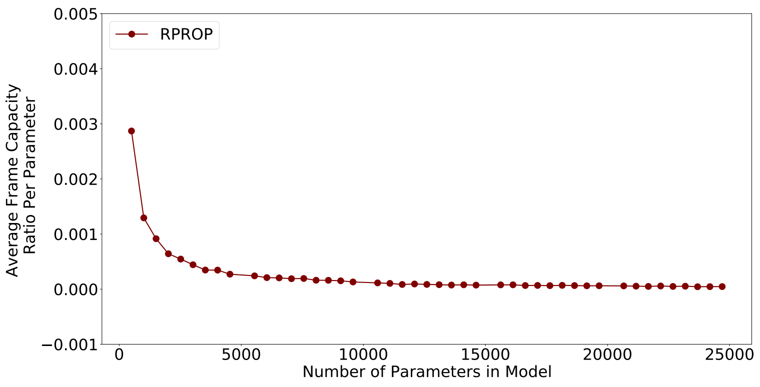

FIGURE 7.179: **Effect of increasing the number of parameters on the per parameter capacity of the network using RPROP training algorithm**

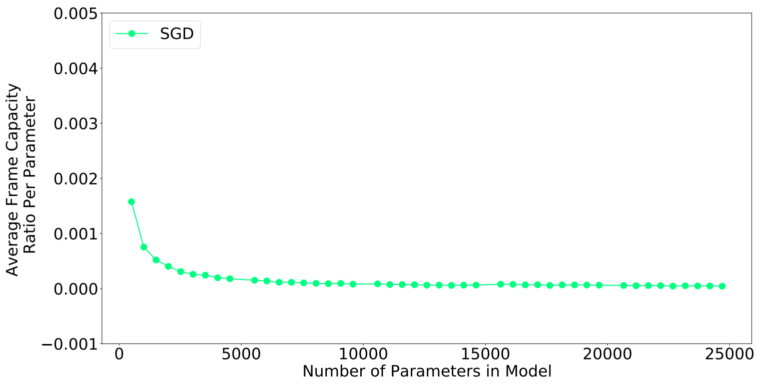

FIGURE 7.180: **Effect of increasing the number of parameters on the per parameter capacity of the network using SGD training algorithm**

Table 7.25 presents the per parameter frame capacity achieved by each RNN using different training algorithms. The highest per parameter frame capacity of 0.0001403 achieved is by the GRU RNN trained using Adam. The bidirectional GRU achieved the highest per parameter frame capacity of 0.0001286 when considering the average overall of the tested training algorithms. The bidirectional Elman RNN performed the worst in each case regardless of which training algorithm was used. Reviewing each RNN reveals that the Adam training algorithm should be used when using the bidirectional Elman, bidirectional Jordan and Elman RNNs as these networks achieved the highest relative frame rate capacity when compared to using other training algorithms. For the bidirectional LSTM and LSTM, RPROP was more effective in training to achieve the highest per parameter frame capacity, achieving a rate of 0.0001425 and 0.0001409 respectively. Lastly, the Jordan RNN had the best performance per parameter when trained using Nesterov for this experiment as opposed to any other training algorithm with a per parameter frame capacity of 0.0000698. On average the Adam training algorithm achieved the highest frame retention ratio across all RNNs of 0.00009449, which makes sense as the Adam training algorithm can make more granular weight updates as a learning rate is maintained for each weight in the network.

	Training Algorithms			
RNNs	Adam	Nesterov	RPROP	SGD
Bidirectional Elman	0.0000390	0.0000226	0.0000251	0.0000206
Bidirectional GRU	0.0001468	0.0001113	0.0001458	0.0001105
Bidirectional Jordan	0.0000545	0.0000472	0.0000479	0.0000458
Bidirectional LSTM	0.0001349	0.0001011	0.0001409	0.0001042
Elman	0.0000478	0.0000270	0.0000308	0.0000289
GRU	0.0001403	0.0001087	0.0001412	0.0001063
Jordan	0.0000572	0.0000698	0.0000595	0.0000678
LSTM	0.0001354	0.0001043	0.0001425	0.0001023

Table 7.25: Per parameter capacity between Number of patterns retained and the number of network parameters using different training algorithms for each RNN

Chapter 8

Conclusion

This chapter serves to summarise findings over all experiments conducted. A summary of the experiments conducted is presented in section 8.1. This is followed by resulting conclusions in section 8.2. After the summary of this work is presented, possible future work identified will be presented in section 8.3.

8.1 Summary

The work set out in this dissertation attempted to understand the retention length and memory capacity of recurrent neural networks. This was done by performing three types of experiments as presented in chapter 6. The first measures the sequence length retention as described in section 6.2.1. The second was constructed to measure the number of patterns which can be retained, the details of which can be found in section 6.2.2. In the last experiment both of the previous experiments were combined, that is, to determine the number of sequences which could be retained as the number of patterns and length of sequences increased.

The first two experiments were formulated as both regression and classification variants. This was done to investigate how formulating the problem statement may affect the retention of sequences and patterns respectively. The last experiment which combined the former, was only formulated as a regression experiment as the results of the first two experiments alluded to RNNs performing better in regression formulated domains. Experiments were continued as long as the RNN's showed increase in performance over a set number of epochs as discussed in the performance chapter 5.

RNN nodes, such as the LSTM and GRU, have more parameters than that of the Elman and Jordan. More so the bidirectional RNNs have double the parameters than the single directional counter parts. In order to conduct a fair comparison, the performance of these RNNs are compared on a per parameter capacity ratio. The details of which is presented in section 6.1.1 of chapter 6. For sequence retention, this ratio is referred to as the per parameter length retention. In the case of the number of patterns, it is referred to as the per parameter pattern capacity. And in the last experiment, this is referred to as the per parameter frame capacity.

In each experiment many different components which could affect the retention of RNNs were explored. First a holistic view of the performance of each RNN was presented. Thereafter the effects of using different activation functions were investigated. This was followed by the effects of increasing the number of hidden layers. And lastly the effects of using different training algorithms were investigated.

The activation functions explored is presented in section 6.1.3 of chapter 6. The functioning of each training algorithm used is extensively discussed in chapter 4.

8.2 Conclusions from experiments

Considering experiment one in which the length of retention in RNNs are investigated, the bidirectional GRU was able to achieve the highest per parameter length capacity for sequence retention overall sub-experiments conducted. Bidirectional layers generally increased the effectiveness of length retention capacity for each RNN. As the number of model parameters increased, the number of sequences retained dropped off in the regression experiments, but slightly increased in the classification experiments. As the number of parameters in the networks increased so did the per parameter capacity decrease. Next the effects of using different activation functions on the length retention capacity of RNNs were investigated. It is found that the RNN-activation function pair which achieved the highest length retention was the bidirectional GRU using the elu activation function. However, the activation function which had the best average performance across RNNs was the tanh activation function. Increasing from one to two hidden layers drastically decreases the per parameter capacity ratio, thereafter the ratio becomes more stable. This indicates that having fewer, but wider layers may lead to more information retained in the networks. When using training algorithms, the Adam training algorithm applied to the bidirectional GRU achieved the best per parameter capacity retention of sequences, however the RPROP had the better overall increase in sequence retention.

Moving on to experiment two which focused on investigating how many patterns RNNs are able to retain, the regression variant was also better suited for RNNs to have a larger retention of patterns than the classification experiment formulation. Here, the bidirectional GRU had the best per parameter pattern retention. Generally, the bidirectional layers enhanced per parameter utilisation, as was the case in the first experiment, with the the only exception being the bidirectional Elman RNN. During experiment two, increasing the number of parameters in the models increased the overall number of patterns retained, however still lead to a decrease in per parameter capacity. The softmax activation function had the highest correlation to performance increase as the number of parameters in the network increased. However, when the per parameter capacity is considered, the selu was the most effective activation function. Furthermore, increasing the number of layers increased the capacity of the RNNs, however the average number of patterns retained dips between using two to three hidden layers and thereafter picks up again for multiple RNNs. Increasing the number of layers, decreases the overall per parameter pattern capacity. Moving on to analysing how the training algorithms affected this experiment, it was found that the Adam training algorithm lead to the greatest per parameter pattern capacity.

Experiment three, which combined the previous experiments, had similar behaviour to the behaviour observed in experiment one and two. The bidirectional GRU achieved the highest frame capacity per parameter overall. However, bidirectional layers did not improve all RNNs as in previous experiments, as only the GRU's per parameter frame capacity increased. Increasing the number of parameters did improve the frame capacity of the RNNs on average. Inspecting the effects of activation functions it was clear that the selu activation function lead to the highest average per parameter frame capacity over all RNNs. However, the LSTM using the relu activation function lead to the highest average per parameter capacity compared to all other tested activation functions. When investigating the effects of increasing the number of hidden layers, the frame retention capability of the GRU increased the most from increasing the network depth. Lastly, the Adam training algorithm resulted in the highest number of frames retained, however had a higher variance than the other algorithms.

8.3 Future work

Upon inspection of the experiments a few behaviours were discovered which could be explored further. One of which is the dip in performance when transitioning from using two hidden layers to using three hidden layers, only for performance to increase when using four hidden layers. Nakkiran et al. suggest the phenomenon observed here might be due to deep double descent in which neural networks first decrease in performance [36]. This may be problem dependent, however further investigation is needed.

Another candidate for further research is to ascertain the reasons for the per parameter frame capacity not increasing for most RNNs when using bidirectional layers. In both experiments one (sequence length of retention) and two (number of patterns retained), bidirectional layers were advantageous, thus the expectancy is for frame retention, bidirectional layers should increase frame capacity.

Lastly, investigate how changing the internal activation functions inside the LSTM and GRU nodes would affect the performance of these networks. Particularly, the sigmoid functions used within.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[3] Monica Adya and Fred Collopy. How effective are neural networks at forecasting and prediction? a review and evaluation. *Journal of forecasting*, 17(5-6):481–495, 1998.

[4] Yoshua Bengio, Nicolas Boulanger-Lewandowski, and Razvan Pascanu. Advances in optimizing recurrent networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8624–8628. IEEE, 2013.

[5] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[6] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: A cpu and gpu math compiler in python. In *Proc. 9th Python in Science Conference*, volume 1, pages 3–10, 2010.

[7] Sergio Bermejo and Joan Cabestany. Oriented principal component analysis for large margin classifiers. *Neural Networks*, 14(10):1447–1461, 2001.

[8] N Brunel, J-P Nadal, and G Toulouse. Information capacity of a perceptron. *Journal of Physics A: Mathematical and General*, 25(19):5017, 1992.

[9] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[10] François Chollet et al. Keras. https://keras.io, 2015.

[11] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[12] Jasmine Collins, Jascha Sohl-Dickstein, and David Sussillo. Capacity and trainability in recurrent neural networks. *arXiv preprint arXiv:1611.09913*, 2016.

[13] Thomas M Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE transactions on electronic computers*, 1(3):326–334, 1965.

[14] Thomas M Cover. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE transactions on electronic computers*, 1(3):326–334, 1965.

[15] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240. ACM, 2006.

[16] Yadolah Dodge. *Spearman Rank Correlation Coefficient*, pages 502–505. Springer New York, New York, NY, 2008.

[17] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

[18] Andries P. Engelbrecht. *Computational Intelligence: An Introduction*, volume 1 of *1*, chapter 3 Supervised Learning Neural Networks, pages 27–42. John Wiley and Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex PO19 8SQ, England, 2 edition, 2007.

[19] Elizabeth Gardner. Maximum storage capacity in neural networks. *EPL (Europhysics Letters)*, 4(4):481, 1987.

[20] Felix A Gers, Douglas Eck, and Jürgen Schmidhuber. Applying lstm to time series predictable through time-window approaches. In *Neural Nets WIRN Vietri-01*, pages 193–200. Springer, 2002.

[21] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural Computation*, 1999.

[22] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1, chapter 6.5.1 Computational Graphs, pages 204–205. MIT press Cambridge, 2016.

[23] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376. ACM, 2006.

[24] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, page 14, 2012.

[25] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91(1), 1991.

[26] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[27] Sepp Hochreiter and Jürgen Schmidhuber. Lstm can solve hard long time lag problems. In *Advances in neural information processing systems*, pages 473–479, 1997.

[28] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[29] Guang-Bin Huang. Learning capability and storage capacity of two-hidden-layer feedforward networks. *IEEE Transactions on Neural Networks*, 14(2):274–281, 2003.

[30] Aleksey Grigorievitch Ivakhnenko. The group method of data handling – a rival of the method of stochastic approximation. *Soviet Automatic Control*, 13(3):43–55, 1968.

[31] Kevin Jarrett, Koray Kavukcuoglu, Yann LeCun, et al. What is the best multi-stage architecture for object recognition? In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2146–2153. IEEE, 2009.

[32] Michael I Jordan. Serial order: A parallel distributed processing approach. *Advances in psychology*, 121:471–495, 1997.

[33] Ujjwal Karn. A quick introduction to neural networks. https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/, 2016.

[34] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[35] Jorge F Mejias and Joaquín J Torres. Maximum memory capacity on neural networks with short-term synaptic depression and facilitation. *Neural computation*, 21(3):851–871, 2009.

[36] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. *arXiv preprint arXiv:1912.02292*, 2019.

[37] Yurii Nesterov. Introductory lectures on convex programming volume i: Basic course. *Lecture notes*, 1998.

[38] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

[39] Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster back-propagation learning: The rprop algorithm. In *Neural Networks, 1993., IEEE International Conference on*, pages 586–591. IEEE, 1993.

[40] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[41] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[42] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[43] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[44] Hava T Siegelmann and Eduardo D Sontag. On the computational power of neural nets. *Journal of computer and system sciences*, 50(1):132–150, 1995.

[45] Akash Singh. Anomaly detection for temporal data using long short-term memory (lstm), 2017.

[46] Kai Ming Ting. *Confusion Matrix*, pages 260–260. Springer US, Boston, MA, 2017.

[47] Paul Werbos. Beyond regression:" new tools for prediction and analysis in the behavioral sciences. *Ph. D. dissertation, Harvard University*, 1974.

[48] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[49] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.