WORCESTER POLYTECHNIC INSTITUTE

MASTER'S THESIS

Exploring Neural Network Structure through Iterative Neural Networks: Connections to Dynamical Systems

Author:
Quincy HERSHEY

Advisor:
Professor Randy PAFFENROTH
Reader:
Professor Seyed ZEKAVAT

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Data Science.

Spring 2023

Advisor	Date
Reader	Date

Abstract

Quincy HERSHEY

Exploring Neural Network Structure through Iterative Neural Networks: Connections to Dynamical Systems

The study of neural networks often places a heavy emphasis on structural hyperparameters with the underlying architecture denoted as a series of nested functions within a rigid architecture. The structure and depth of a given network are typically specified as hyperparameters including the number and size of layers within a given model in a task dependent manner. In the process of exploring network architectures this thesis also examines the roles and relationship of network depth and sparsity in determining performance. The concept of aspect ratio is introduced, as a central feature governing the relationship between the depth and width of the combined network weight space. Consideration is given toward whether network structure in more traditional architectures serves as a proxy for the roles of depth, aspect ratio and sparsity. After demonstrating the traditional feed forward multi-layer perceptron (MLP) neural network architecture to be a special case of a recurrent neural network (RNN), the influence of these factors on network performance is examined from the alternate perspective of an RNN architecture. Recurrent neural networks contain clear commonalities with dynamical systems theory where iterative structures are prominently used in place of nested functions. While capable of replicating the traditional MLP architecture as a specific case, the RNN exists as a more generalized format of neural networks. Throughout this research, the problem sets employed in benchmarking model performance include typical MNIST based visual performance tasks as well as derived datasets representative of anomaly detection sequence and memory tasks. Among these comparisons, the relative performance of Long Short-Term Memory (LSTM) and MLP class models are benchmarked versus comparatively less-structured sparse RNN models. Lastly, sparse RNN's are considered as a possible mechanism for gauging problem set difficulty.

Acknowledgements

I would like to thank my advisor, Dr. Randy Paffenroth for passing on his bright interest in the arena of data science to his students. Additionally, I am thankful for the culture and resources made available through Worcester Polytechnic Institute without which much of this research would not have been possible. This research was performed using computational resources supported by the Academic & Research Computing group at Worcester Polytechnic Institute.

Contents

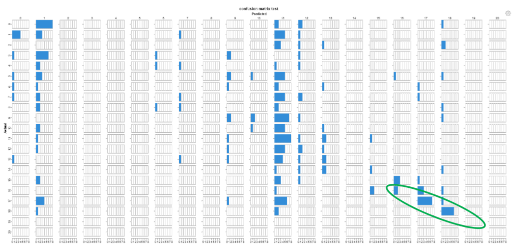
List of Figures

Executive Summary

This body of research utilizes the concept of neural networks as iterative mathematical equations to study the underlying principals and applied assumptions embodied in traditional neural network architectures. The key idea is that Recurrent Neural Networks (RNN) architectures are more general than widely understood. In particular as we show here, every Multi-Layer Perceptron can be rewritten as a RNN. Therefore, Multi-Layer Perceptrons are a subset of RNN's. These ideas result is a more universal approach governing neural network architecture and hyperparameter selection aligned with an alternate set of recommended metrics.

Through the use of mathematical derivations this study first demonstrates equivalence between RNN architectures presented as iterative functions and the typical nested representations used for standard feed forward networks. These findings are then explored with the goal of more deeply understanding the drivers behind model performance. Using MNIST derived data sets to construct experiments, model performance is benchmarked both against other popular model architectures and against various configurations of RNN's. Early rounds of experiments generate surprising results running contrary to common wisdom. For instance, randomly arranged parameters within a sparse matrix demonstrate no distinguishable difference in model performance over the use of neatly arranged layers. The findings seem to imply the specification of model structure through the number and dimension of layers is an artificial construct carrying little direct impact that attempts to indirectly approximate other implied model characteristics.

Building on these findings, the thesis explores alternate characteristics that may serve as better hyperparameters in governing model performance particularly when deployed among sparse RNN's. The research and experiments then examine model expressiveness and stability through the interrelated concepts of total trainable parameters, model depth and sparsity with each playing a role in determining model performance. Model sparsity emerges as a significant factor in performance while greatly enhancing stability and expressiveness at significantly reduced parameter counts. The aspect ratio of the weight space, reflecting the relationship between the depth and width of the weight space is introduced as a central feature governing model performance with direct ties to the iterative nature of RNN's. Beyond serving as a useful hyperparameter in defining model architectures, this feature combines with model sparsity to offer additional applications in quantifying and comparing task difficulty.


(A) LSTM performance on random anomaly detection series with length 19.

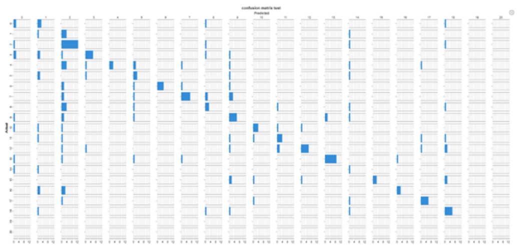

(B) Sparse RNN performance on random anomaly detection sequence of length 19.

FIGURE 1: Side-by-side comparison of confusion matrices contrasting performance of LSTM and sparse RNN networks on the random anomaly detection problem with sequence length 19. The LSTM network routinely demonstrated a clear recency bias, showing relative improvement versus itself on anomalies occurring later in the sequence. However, the sparse RNN models demonstrated a significant performance advantage versus LSTM networks with similar numbers of total trainable parameters.

Sparsely arranged RNN's perform well across widely varied experiments when benchmarked against popular classes of competing model architectures. Through robust performance across a wide array of tasks, RNN's demonstrate the benefit of sparsity and the inherent flexibility in efficient allocation of weights. Using several problems, RNN's consistently beat or matched competing peer LSTM and MLP networks across a variety of tasks. In one example of an anomaly detection task, Figure 1 demonstrates a sparse RNN significantly outperforming a comparable LSTM network with a similar number of trainable parameters.

While recurrent neural networks are widely understood to be iterative network architectures, the recognition that MLP's exist as a subset of RNN's carries secondary implications. By extending the generalized iterative approach to the wider body of neural network architectures there are opportunities for knowledge transfer from neighboring domains. By example such iterative equations are a central component of dynamic systems theory, a well established field of study suited to complex systems often associated with feedback mechanisms and other nonlinear behavior. These systems commonly feature sequentially dependent states, each evolved within a closed system such that knowledge of an initial state S^0 allows prediction of states at any subsequent point in time as are often encountered within classical physics. Famous examples include the Mandelbrot Set which joins dynamical systems with fractal geometry [17]. Within the field of dynamical systems, those systems with particularly high sensitivity to S^0 find that small changes to the initial state may result in wildly varying outcomes or similarly chaotic behavior as is the case with a double pendulum in Figure 2. This area of specialization, referred to as chaos theory, shares many commonalities with neural neural networks including feedback loops, interconnectedness, self-regulation and patternistic behavior which often emerge in the form of exploding gradients [25, 27]. These topics provide a wide surface area for exploration and motivate much of the research in this area seeking to bridge the understanding between these two fields.

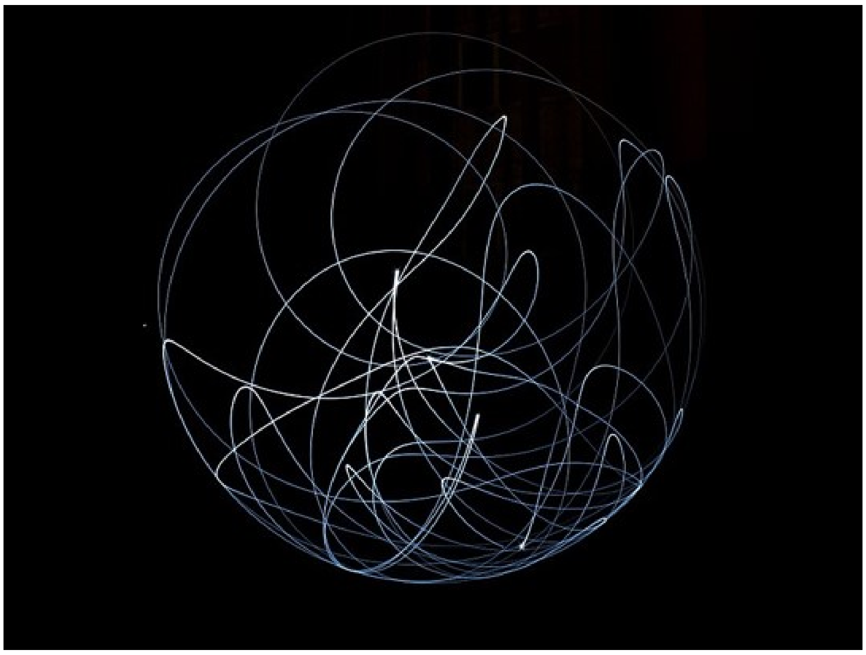


FIGURE 2: Chaos systems at work: a double pendulum's path traced under long exposure [36]. The behavior of a double pendulum is prone to wild swings and erratic moves predicated on a fixed behavior from a known initial state.

The combination of these findings redirects the understanding of neural network architecture and hyperparameter selection towards a more universal approach while recommending alternative metrics for use in characterizing network structure. These findings run counter to common intuition and open a path for further research into the implications of generalized architectures and randomly assigned sparse weight spaces towards model behavior.

Chapter 1

Introduction

1.1 Overview

Initially developed in 1958, neural networks [20] seek to approximate complex functions primarily in nonlinearly separable solution spaces. These networks combine nonlinear behavior with a highly parameterized network requiring few underlying assumptions while representing a wide array of functions. To this end, George Cybenko's Universal Approximation Theorem states a single layer feedforward network of arbitrary width can represent any function [8]. This broad implied capability of neural networks coupled with the need for few underlying assumptions has seen them successfully deployed across a wide array of problems with widespread adoption supported by increasing availability of high powered and optimized hardware resources. As the availability of high-powered computation resources has increased, network depth and breadth have risen alongside advancements in overall network complexity and structure while exploration of neural network architecture continues to occasionally yield unexpected outcomes. Within this context traditional feed forward multi layer perceptron (MLP) architectures are typically presented as distinct from recurrent neural network (RNN) architectures which have a reputation of being difficult to train [23]. Contrary to this perspective, this research presents RNN's as a generalized representation of neural networks within which MLP's exist as a special case suggesting alternate factors may play a role.

A central component of training and deploying neural networks against problem sets involves the selection and specification of hyperparameters governing model architecture and other core features. Examples of common hyperparameters include learning rate, batch size, layers, the overarching structure of layers and model size [2] often measured by trainable parameters. While multiple approaches for hyperparamater selection and tuning exist [6, 13, 22], they are often time consuming for larger models and involve some process of testing combinations factors which rise exponentially as more hyperparameters are considered. Furthermore, the sensitivity of models to these factors can often cause instability within both model training and testing adding to the challenges encountered in this process [35]. Several characteristics such as model depth and model sparsity are presented which significantly improve the stability of model performance across a wider array of hyperparameters and reduce or eliminate performance differentials.

Sparse RNN's are shown to exhibit increased performance and training stability across a wider array of conditions and with reduced parameterization. At its core, this approach allows comparable model performance with total trainable parameters reduced by more than a full order of magnitude in some cases while maintaining the ability to represent complex functions. These findings echo similar findings from recent research. Sparsity has been presented as a technique to reduce computational and memory needs for neural networks, while increasing ease of deployment across devices [21]. Commonly utilized methods involve either explicit block pruning methods or lasso regularization [9, 21] applied through training with findings showing ranges from 80% to more than 90% sparsity resulting in minimal loss of accuracy. Recent research has also shown that contrary to general belief the approach of training sparse RNN's from scratch may generate clear performance advantages over dense networks [14]. As a result, aside from solely improving parameterization efficiency, this approach has been demonstrated to be well suited for complex systems with unknown dynamics such as hurricane topography [19] which carry strong ties to the study of dynamical systems and chaos theory.

1.2 Summary

In Section 2 several core topics are introduced which form a fundamental basis of understanding for the following research. This background begins with a cursory introduction of common network architectures including multi-layer perceptrons (MLP) shown in Section 2.1.1 which are useful across simple categorization and regressing tasks. From there iterative network architectures are introduced by summarizing recurrent neural networks (RNN) in Section 2.1.2. The concepts surrounding iterative network architectures are introduced as a generalized framework inherently capable of replicating both MLP and RNN architectures. Iterative neural networks are presented as a useful framework in conceptualizing the implied differences and assumptions of both model architectures and are demonstrated to be mathematically equivalent to both architectures under various constraints in Sections 2.1.3 and 2.1.4. Following this, long short-term memory (LSTM) networks are summarized in Section 2.1.6 which serve as a performance benchmark as a commonly deployed architecture against sequential data sets. The underlying data sets and experiments are introduced in Section 2.3 before moving into results in Chapter 3 and concluding in Section 4.

Chapter 2

Methodology

2.1 Background regarding neural networks

Several fundamental neural network architectures are introduced in the following sections and later used to explore performance across various problem sets. The iterative nature of Recurrent Neural Network (RNN) architectures is presented as a generalized neural network architecture with roots in dynamical systems theory [1, 32]. Moreover, traditional feed forward multi-layer perception networks may be represented by the RNN's as a special case implemented through constraints regarding the arrangement of trainable weights. An overview is provided of the standard long short-term memory (LSTM) network [37] as a common architecture deployed over sequential data sets and therefore a natural choice for performance benchmarking [5].

2.1.1 Multilayer perceptrons

Neural networks such as the traditional feed forward multilayer perceptron [20] architecture can be defined as a set of nested functions as denoted in Equation 2.3. In this representation, each layer i of the network and accompanying non-linear activation function [26] is represented as function $f^i(x)$, with the linear output a of each layer feeding through a nonlinear activation function σ. The output of this function then forms the hidden layer input h of the subsequent function with the exception of the final layer of weights which provides model output y. The example provided in Figure 2.3 represents a typical four-layer network. Within this feed forward architecture, each function $f(x)$ can be described as in Figure 2.1 as a combination of the dot product of the input x and j weight vectors w of length d with j dependent upon the output dimensionality of $f(x)$ prior to adding a bias b and applying the activation function σ.

$$f_j^i(x) = \sigma(w_j^i x + b_j^i) \tag{2.1}$$

$$f^3(f^2(f^1(f^0(x)))) \tag{2.2}$$

$$f^3 \circ f^2 \circ f^1 \circ f^0(x) \tag{2.3}$$

In the interest of emphasizing the connection to iterative maps, the formulas can be further simplified into matrix notation. This is performed by arranging the input data x into a matrix X where $X \in \mathbb{R}^{d\times n}$, n is the number of input examples combined within a single matrix and d is the dimension of x. Afterwards, attach a single vector of 1's with dimension $1 \times n$ to the base of the input matrix X so that the dimension p of X is the initial input dimension d increased by 1 as demonstrated in Figure 2.4

where $X \in \mathbb{R}^{p \times n}$. By doing so, the weight vector w and bias b may be concatenated into a single matrix W where the final column contains the bias as demonstrated in Figure 2.5. This practice of attaching a vector of 1's to the base of the next layer's input matrix and concatenating the weights and biases into a single matrix occurs at each layer of the network. The end result allows the overall notation to be simplified into the commonly used matrix notation shown in Equation 2.6.

$$X = \begin{bmatrix} x_{00} & \cdots & x_{0n} \\ \vdots & \ddots & \vdots \\ x_{d0} & \cdots & x_{dn} \\ 1 & \cdots & 1 \end{bmatrix} \tag{2.4}$$

$$W^i = \begin{bmatrix} w_{00} & \cdots & w_{0d} & b_0 \\ \vdots & \ddots & \vdots & \vdots \\ w_{j0} & \cdots & w_{jd} & b_j \end{bmatrix} \tag{2.5}$$

$$f^i(X) = \sigma(W^i X) \tag{2.6}$$

The fundamental components of a feed-forward multilayer perceptron (MLP) [20] form a starting point from which network structure may be specified through hyperparameter selection as shown in Figure 2.1 illustrating the case of a simple MLP with output dimension 1. The field has expanded from this initial point to include many neural network architectures with unique design considerations. Each evolution in neural network design offers complex structures and model characteristics allowing widespread adaptability to satisfy complex problems across many domains.

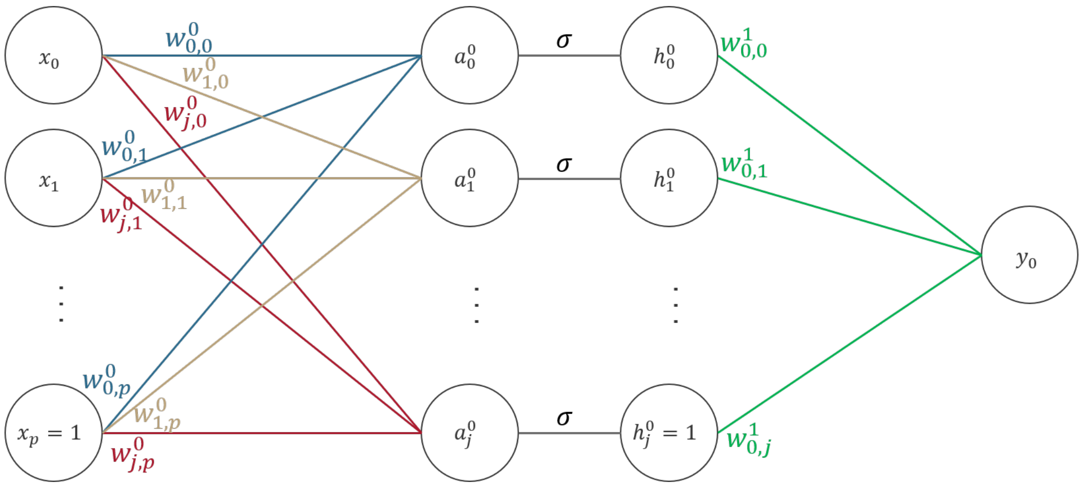


FIGURE 2.1: Two layer diagram of typical MLP architecture.

While the standard MLP neural network has proven to be a versatile and strong performer for many estimation and classification tasks, it suffers from the limitation of statelessness in that it retains no stored memory after each input example of the instances it has processed before. For sequential data sets which take place either as part of a greater whole or which hold some relationship with the surrounding members in a given sequence, memory is clearly a desirable or even necessary trait. For these cases there exist many architectures for analyzing data in series, among which are the Recurrent Neural Network (RNN) and the Long Short-Term Memory Network (LSTM) which are discussed next.

2.1.2 Recurrent neural networks

The Recurrent Neural Network (RNN) [30, 33] architecture was developed to overcome the limitations of MLP networks with respect to sequential data sets. The RNN achieves this by iteratively applying shared parameters over each item in a sequence and incorporating a stored hidden state H as outlined in equation 2.7. Notation follows the mechanisms from equations 2.1, 2.4, 2.5. The first hidden state is generally but not necessarily initialized as a zero matrix for H^0. For each item in an input sequence X, a weight matrix W^i is multiplied by the input matrix X^t at step t while another weight matrix W^h is multiplied by the hidden state H^{t-1} from the previous step $t-1$ with some nonlinearity function (often a ReLU or tanh function) to the output. The resulting output H^t then serves as the next hidden state alongside the next item in the sequence X^{t+1} using the same weight matrices W^h and W^i, respectively. When the last step in the sequence has been reached, the final hidden state serves as the output.

$$H^t = ReLU(W^i X^t + W^h H^{t-1}) \tag{2.7}$$

By concatenated both weight matrices W^h and W^i as simply W and concatenating X^t and H^{t-1} as a single input matrix, $\bar{X}^t$ the resulting diagram is illustrated in Figure 2.2. Through the use of shared weights, the RNN appears to unroll across the sequence steps with no theoretical limitation on sequence length paired with the ability to process varying length sequences. The simple representation we've discussed portrays the RNN as a single layer network processing data in a single direction moving forward through the sequence, however in practice they may be built as a series of recurrent layers feeding one another sequentially or processing data series bidirectionally. Despite the technical ability for use with extremely long input data sequences, in practice while backpropagating the errors over many iterations RNN's are known to become prone to either exploding or vanishing gradients that limit the network's memory over long sequences. In these cases, the process of calculating gradients for repeated operations and nonlinearity functions across many steps may cause the magnitude of gradient to either vanish or explode causing instability. These effects are reduced through the use of ReLU as the choice of nonlinearity function, given that for positive values the gradient is unchanged. Nonetheless, these general limitations inspired the creation of long short-term memory networks covered in 2.1.6.

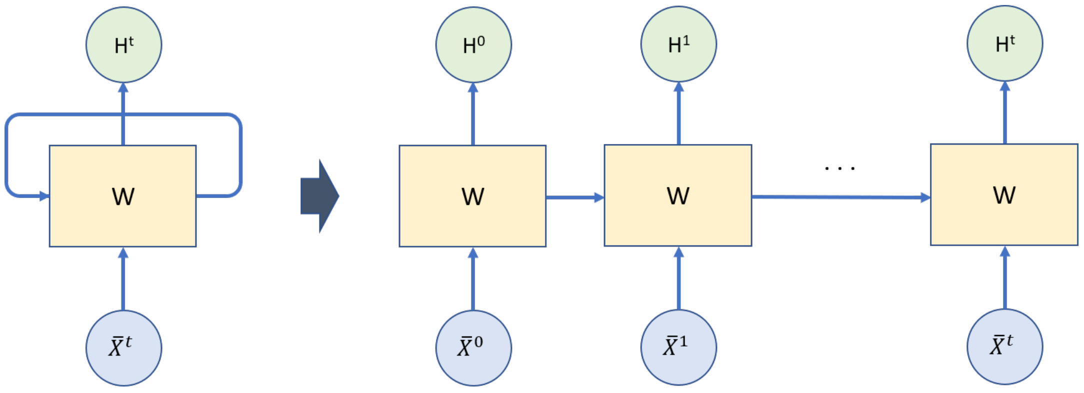


FIGURE 2.2: RNN diagram across steps of a sequence.

Despite the emergence of alternatives, RNN networks remain a common choice for sequential analysis given their ease of use, general effectiveness and intuitive structure. Such networks can either be used as stand alone neural networks or as building blocks within a larger network, commonly with a linear feed forward layer projecting the output as in Section 2.1.1. As it emerges in 2.1.4, the typical RNN may be constructed in a manner so as to be directly equivalent to a broader class referred to as iterative neural networks and defined in the following section.

2.1.3 Iterative networks as generalized MLPs

The roots of iterative representations of neural networks are grounded in the tenets of dynamical systems theory. Dynamical systems theory encompasses a well-established field of mathematical study supported by an extensive body of research and theory [28]. The area has proven well suited to complex systems including characteristics such as feedback and nonlinear behavior with famous examples including the Mandelbrot Set [18] and fractal geometry [16]. Core features of dynamic systems include functional structures which iterate upon themselves [3, 34] to form feedback loops as denoted in Equation 2.8 below.

$$f \circ f \circ f \circ f(X) \tag{2.8}$$

In this section, MLP's are demonstrated to be another example of dynamical systems which may be expressed through iterative notation. The visual similarities between the two functional structures from Equation 2.3 and Equation 2.8 may be approached notationally by demonstrating equivalence. Additionally these models can be implemented using popular deep learning software environments in a manner that allows further study of iterative neural networks. Within the context of this study, **Iterative Neural Network (INN)** architectures are defined as a broader class of networks in which the parameters are applied recursively against the input over multiple repetitions. INN's are presented as a more generalized concept versus RNN's which typically feature dense parameter matrices, whereas the parameters of INN's may be arranged in a wide array of configurations including sparse configurations and self contained layers. The broader inspiration in demonstrating equivalence between INN's and MLP's would be the potential to leverage and benefit from cross application of dynamic systems principals and theory. More directly, the restatement of traditional neural network architectures in an iterative format yields several immediate questions with clear implications to our understanding of the principals governing neural network structures.

As a first step, the problem may be approached by notationally restating the traditional neural network into a mathematically equivalent iterative function. Beginning with a basic two-layer MLP feed forward network and using the notational structure from Equation 2.3, the network can be depicted as the series of nested functions in Equation 2.8. Using the structure outlined in Section 2.1.1 applied to the two-layer case, we begin with the overall architecture of the two-layer MLP network as demonstrated in Figure 2.3, below.

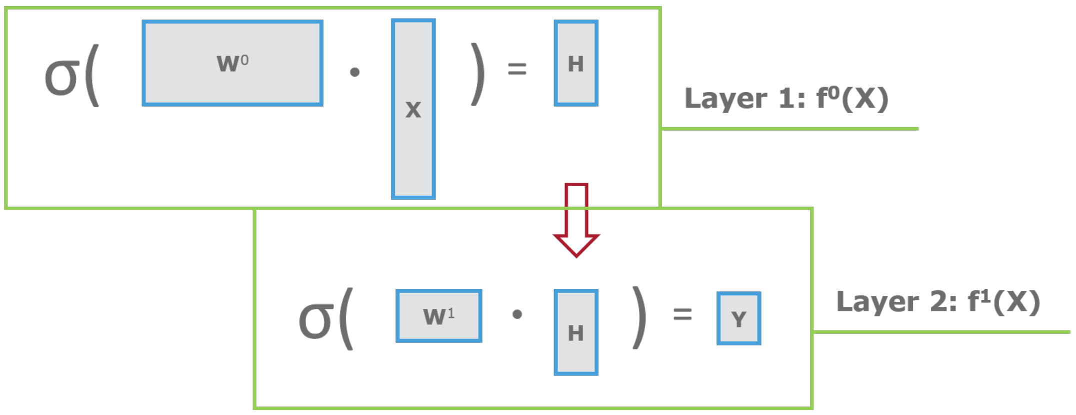


FIGURE 2.3: Standard representation of a two-layer MLP.

$$f^1 \circ f^0(X) = \sigma(W^1\sigma(W^0X)) = Y \tag{2.9}$$

Together, these component matrices will be used to compile a single iterative network function represented by the square matrix $f(X)$. To begin, the component matrices X, H and Y are vertically concatenated together forming a single expanded input matrix $\bar{X}$ where H and Y are each initialized as zero matrices. Using the stated dimensions from Section 2.1.1, $X \in \mathbb{R}^{p\times n}$ with input dimension p over n examples. Similarly, matrix H shares the dimension of the hidden layer h over n examples and matrix Y has output dimension y across n examples. The resulting $\bar{X}$ matrix will have dimension $(p+h+y)\times n$. At this point a single square zero matrix of dimension $(p+h+y)\times(p+h+y)$ is initialized representing the iterative weight matrix $f(X)$, constructed in block-wise fashion using untrainable zero matrices with several exceptions. The upper leftmost component matrix is initialized as an untrainable square identity matrix I with each side matching the input dimension p. Additionally, trainable weight matrices W^0 and W^1 are inserted as demonstrated in Figure 2.4. Finally, the matrix is wrapped in the relevant activation function as demonstrated in Figure 2.4 to form a combined representation of an iterative $f(X)$. The outcome is a single square parent matrix encompassing both weights matrices joined with regions composed of untrainable zero matrices wrapped in a pointwise nonlinearity function. This specific case represents a single encompassing iterative functional representation of the two-layer MLP example from Figure 2.3. This square matrix $f(X)$ is multiplied twice by the concatenated input matrix $\bar{X}$ reflecting a single iteration per layer of weights. The resulting output is mathematically identical to the initial feed forward MLP that served as a starting point. Under this iterative representation, the region of $\bar{X}$ denoted as the Y matrix continues to represent the output of the neural network just as the region denoted as H continues to represent the output of a hidden layer.

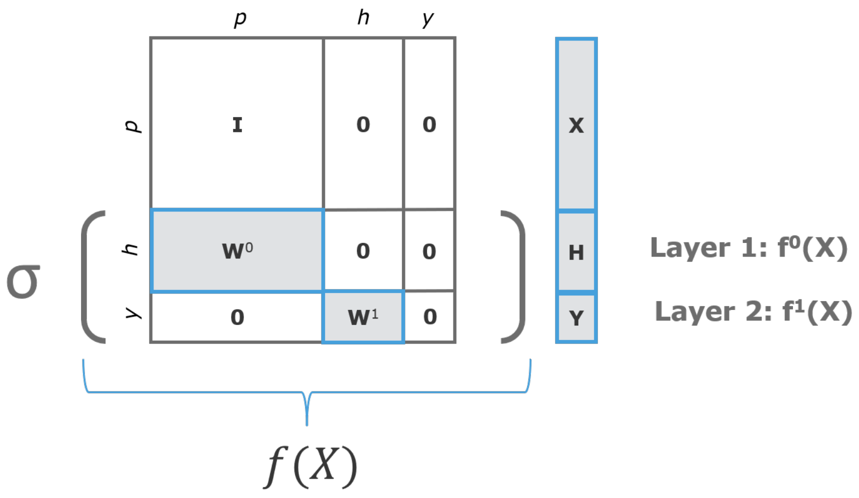


FIGURE 2.4: Iterative Neural Network (INN) representation of a two-layer MLP.

Following two iterative applications (once per layer) of this iterative neural network $f(\bar{X})$, the resulting output is outlined in Equation 2.10. It should be immediately clear that the resulting output is mathematically identical to the output denoted in Equation 2.9 notationally bridging the perceived differnces between Figure 2.3 and Figure 2.4.

$$f(f(\bar{X})) = \sigma(W^1\sigma(W^0X)) = Y \tag{2.10}$$

The INN representation of the neural network makes explicit the untrainable zero matrices surrounding the weight matrices which are implied in the traditional feed forward MLP and other typical feed forward networks. Together, the regions of the weight matrix horizontally aligned with the trainable weights as depicted in Figure 2.5 form the **weight space**. Each parameter within the weight space shares the common characteristic of being subject to pointwise application of the nonlinearity function σ, with the potential to be parameterized and function as a trainable weight across broader applications of the iterative function in alternative configurations.

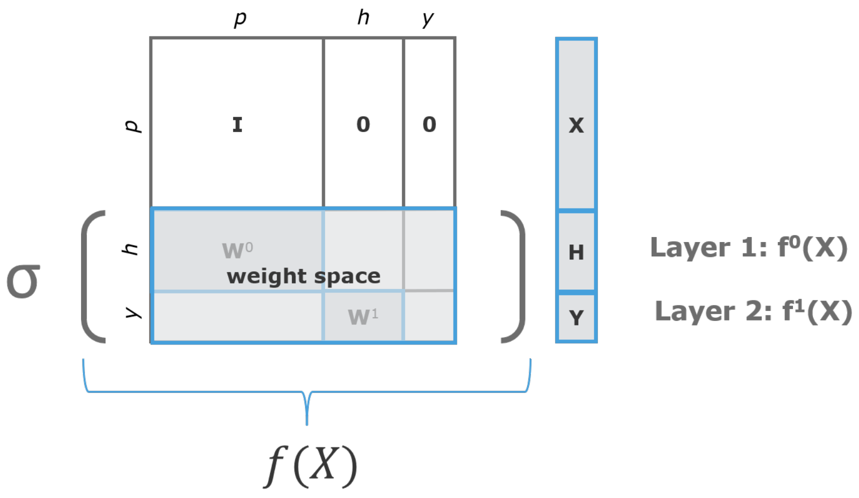


FIGURE 2.5: INN representation of a two-layer MLP with the structure of the weight space expanded to include all regions horizontally adjacent to the weight matrices W^0 and W^1.

Having established the ability for the iterative matrix to replicate a traditional feed forward MLP and achieve mathematical equivalence an additional consideration lies with implementation. The identity matrix I and zero matrices located above the weight space within the iterative matrix in Figure 2.5 serve the purpose of returning the input component matrix X to the larger matrix $\bar{X}$ made possible by the absence of pointwise nonlinearity σ. An alternate but equivalent approach to implementation removes portions of the iterative matrix lying above the weight space as depicted in Figure 2.6. Instead the function of the deleted identity matrix is performed prior to each iteration by inserting a copy of the initial input component matrix X into the matrix $\bar{X}$. Approaching the network in this manner recasts the input X as a sequence of identical steps, with each step comprised of duplicate copies of X. Rather than specifying iterations independently of the input, the network now iterates over each step X^t within the sequence X with one occurrence per layer while the output remains the same. Sequential notation is adopted across the model to differentiate output at each time step t resulting in $\bar{X}^t$ composed of X^t, H^t and Y^t.

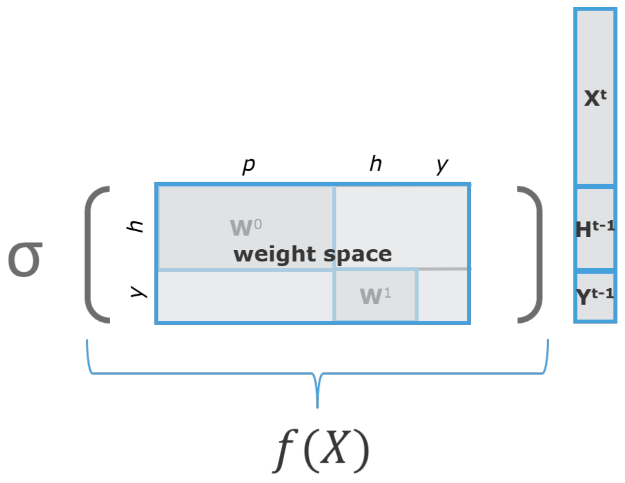


FIGURE 2.6: Alternate implementation of iterative network replicating a two-layer MLP truncated to include only the weight space with the input component matrix X inserted into the larger input matrix $\bar{X}$ prior to each iteration.

Within the weight space, the design choice to define the matrices surrounding each MLP layer as untrainable zero matrices is an implied decision and forms an unnecessary constraint. This recognition gives rise to questioning whether other and perhaps better alternatives exist and what principals may govern optimal definition of architectures within those regions. As one example, initializing some combination of the matrices along the diagonal of the iterative matrix with untrainable identity matrices allows the function $f(X)$ to retain persistent memory of the data contained in each region of $\bar{X}$ for each iteration of $f(\bar{X})$. The immediate impact of this change would appear to be minor within the scope of the overall network, but creates new capabilities and allowing the network to perform when iterations exceed the number of layers. By retaining memory at each step, data from each prior iteration continues to factor into model training and output. However, this change is simply one of many possibilities with the matrices capable of being initialized in any number of combinations and configurations. Importantly, the choice of matrix trainability within the weight space represents yet another hyperparameter to be optimized and studied. This same line of reasoning can be extended to both the zero matrices lying above the diagonal drawing information backwards through the model and the zero matrices below the diagonal which can serve as skip connections. These characteristics underscore the inherent flexibility of the INN architecture, presented here as a more generalized version of the MLP. Taking the idea a step further, the parent matrix $f(X)$ may be arbitrarily subdivided with granularity as fine as individual parameters allowing detailed specification of network structure. This capability enables the INN architecture to replicate other model types while also exploring the impact of parameterization on model performance.

2.1.4 Iterative networks as generalized RNNs

As stated in Section 2.1.3 the Iterative Neural Network (INN) representation of an MLP contains several design choices implied by the feed forward architecture in a multi layered format. Continuing from the example in Figure 2.6 these constraints are removed in several stages. By first relaxing the structural constraints associated with this special case and allowing the weight space to become a fully trainable matrix of parameters, the network assumes a more generalized form as demonstrated in Figure 2.7. At this point, the generalized model has become a single dense layer of parameters with the distinction between H^t and Y^t becoming a purely notational construct, with both folded into a single input/output matrix H^t as also demonstrated in Figure 2.7.

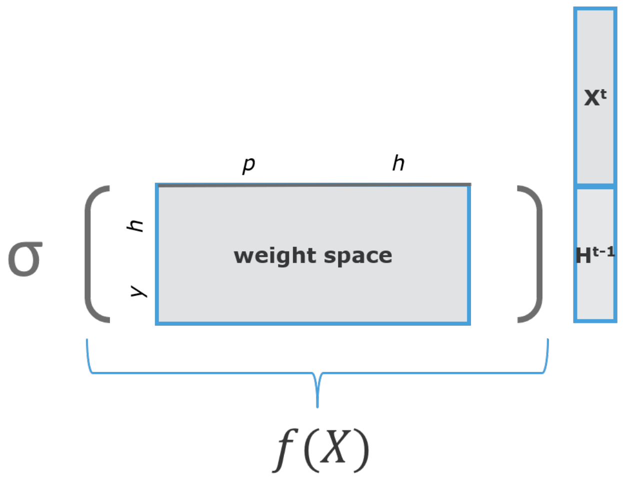


FIGURE 2.7: Structural constraints are relaxed for the INN representation of an MLP from Figure 2.6 in Section 2.1.3, allowing it to assume a more generalized form. The weight space is fully parameterized to form a single dense matrix fully comprised of trainable parameters while variables H^t and Y^t are collapsed into a single input/output matrix labeled H^t.

Taking the model in Figure 2.7 and modifying the notation, the weight space may be broken into two notational matrices: W^i to be multiplied by X^t and W^h to be multiplied by H^{t-1}, respectively. The nonlinear activation function is changed from a traditional sigmoid function to ReLU and the resulting INN architecture shown in Figure 2.8 is now structurally equivalent to the RNN described in Section 2.1.2.

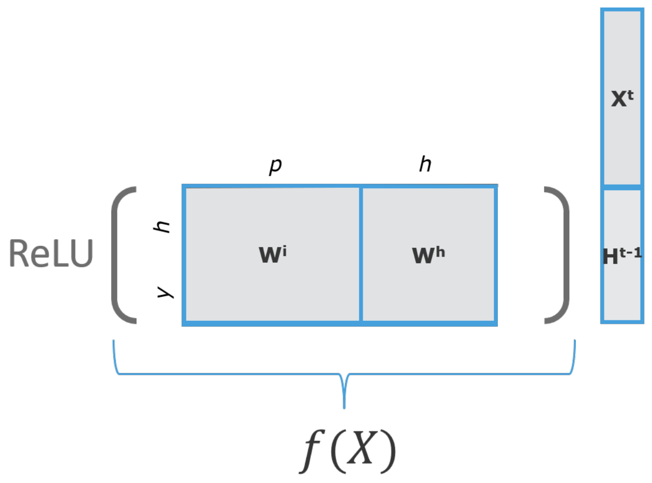


FIGURE 2.8: Notational adjustments are made with the weight space broken into two component matrices: W^i to be multiplied by X^t and W_h to be multiplied by H^{t-1}, respectively. Additionally, the nonlinear activation function is changed to ReLU.

$$H^t = ReLU(W^i X^t + W^h H^{t-1}) \tag{2.11}$$

In the special case from the prior Section 2.1.3 in which an INN was used to replicate an MLP, the final implementation presented the input X as a sequential variable composed of identical steps X^t as a byproduct of applying multiple iterations. To accommodate the INN architecture, the input and output components of X^t, H^t and Y^t were concatenated into a combined matrix $\bar{X}^t$ which was applied across the entire weight space at each iteration t. The end result was that each identical step X^t was inserted into the variable $\bar{X}^t$. The model would then iterate over each step in the sequence, with sequence length corresponding to the desired number of iterations and defaulting to one iteration per layer when replicating an MLP. Within that special case of the MLP implementation, two constraints exist for input sequence X. The first constraint defaults sequence length to a fixed length corresponding to the number of layers in the model. Having removed the notational construct of layers, this constraint is lifted with the length of input sequence X now permitted to vary arbitrarily. The second constraint requires that $\forall X^i, X^j \in X, X^i = X^j$. This constraint is removed as well, allowing the generalized iterative model to accommodate series with sequential input of varying lengths t and varying data points X^t. The resulting sequential model replicates an RNN following equation 2.7 as shown in Section 2.1.2 and reproduced here as equation 2.11.

2.1.5 Iterative vs RNN: differences in implementation

While implementing RNN's as INN's small differences may arise during implementation. These primarily appear with the introduction of a projection layer to create the final output Y. The common approach here is to sequentially apply a linear projection layer over the final hidden output H. Building on the example from Figure 2.8, then the output of equation 2.11 now forms the input of the projection function f^{proj} which uses multiplication with a parameter matrix to alter the dimension of the final output as shown in equation 2.13.

$$Y^t = f^{proj}(H^t) \tag{2.12}$$

$$Y^t = ReLU(W^{1h} ReLU(W^i X^t + W^h H^{t-1})) \tag{2.13}$$

This result can be replicated using an INN architecture as shown in Figure 2.9 where the parameter matrix W^{1i} is constrained to an untrainable zero matrix, creating a projection layer over the hidden output. Just as in the replication of the layered MLP in Figure 2.4, an extra iteration must be performed to reflect one iteration per layer. This effect occurs because within the iterative network structure Y^t is calculated contemporaneously with H^t, with both Y^t and H^t receiving H^{t-1} as an input at time step t. However, as noted in equation 2.12, the sequential application of the projection must receive H^t as an input, thus requiring an additional iteration, resulting in iterating once per layer.

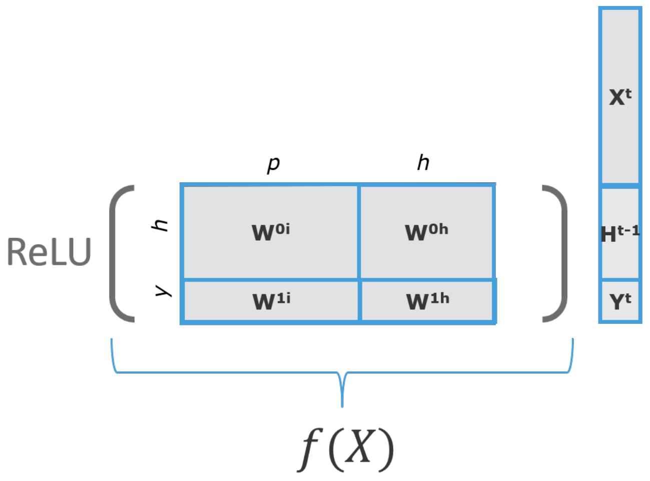


FIGURE 2.9: Following on the example from Figure 2.8, a projection layer is added to the hidden output to reflect equation 2.13. To accurately recreate this output, the parameter matrix W^{1i} is constrained to an untrainable zero matrix.

As noted, the differences that arise in implementation of a projection can be overcome in the example in Figure 2.9 by constraining parameter matrix W^{1i} to an untrainable zero matrix and adding an additional iteration. However, the implementations begin to diverge mildly, yet inextricably when the constraints are relaxed, allowing parameter matrix W^{1i} to assume the form of a trainable sparse matrix as with the other parameter matrices. Under these conditions, the weight space no longer assumes a layered structure and an additional iteration is no longer necessary. By returning to a single iteration, information now flows from the input X^t to output Y^t through the projection in the first iteration as shown in equation 2.14. However, within the INN architecture the output matrix Y^t only receives H^{t-1} from the prior time step as Y^t is calculated contemporaneously with H^t as mentioned previously.

$$Y^t = f^{proj}(X^t, H^{t-1}) \tag{2.14}$$

$$Y^t = ReLU(W^{1i} X^t + W^{1h} H^{t-1}) \tag{2.15}$$

This implementation carries a trade off as the weight space W^{1i} must now compensate for the fact that the output Y^t is no longer receiving the embedded hidden state H^t sequentially at time step t. In practice, the effect remains marginal but worth noting when drawing parallels between RNN's and INN's in cases where projections are employed and the weight space is comprised of sparse matrices without the previously mentioned constraints.

2.1.6 Long short-term memory

The Long Short-Term Memory (LSTM) network [11, 33] addresses a known challenge of RNN architectures in storing information over long sequences using the internal state of the network. Within the typical RNN architecture, the repeated application of the weight matrix and nonlinearity over many time steps often distorts the gradient and thus effective memory of the network. This effect is most pronounced as the steps between two input matrices in a sequential data set expand. The LSTM structure creates the ability for constant error flow within a cell state using multiplicative units called gates 2.16. This structure essentially creates a cell state within the neural network with the ability to bypass interim steps and maintain constant gradient. A typical LSTM network often contains four gates, the input gate I^t, forget gate F^t, cell gate G^t and output gate O^t. Each of the four gates contains two trainable weight matrices and receives the prior hidden state H^{t-1} and input matrix X^t as inputs. The output of each gate then passes through a sigmoid function σ except in the case of G^t, which utilizes a tanh function.

At the start of each step, the forget gate F^t determines whether information should be "forgotten" or cleared from the prior cell state C^{t-1} using a Hadamard product $\odot$. The input gate I^t then serves the purpose of allowing new information to be passed to the cell state. The cell gate G^t then transforms the data that will be additively stored to the cell state. The result of cell gate G^t is then filtered through the results of the input gate I^t using a Hadamard product. Lastly, the output gate O^t governs information outflow from the cell state to the next hidden state H^t through a Hadamard product with the resulting cell state filtered through a tanh function.

A visual diagram outlining the structure of gates in governing addition and removal of data within the cell state is shown in Figure 2.10 using equations 2.16 which reflect the Pytorch [24] implementation [31] of a standard LSTM. The end result is that through the use of gates governing the distinct cell state and a hidden state comparable to that of the RNN, the more sophisticated structure of the LSTM works to address the long-term memory issue. Robust performance from LSTM networks have given rise to the widespread popularity of the LSTM network across sequential applications particularly in the language processing realm. As with the RNN, LSTM networks may be applied bi-directionally across sequences and in layered configurations dependent upon the needs and characteristics of the underlying problem. However, the standard LSTM network as implemented across many neural network architectures does come with several drawbacks. The implementation is relatively structured both with respect to the nonlinearity functions used internally within the network and with respect to the allocation of trainable parameters across various tasks within the network. The structured allocation of weights appears to result in LSTM networks learning less complex representations of the underlying data in comparison to networks with more inherent freedom in allocating weights towards the problem set such as the typical RNN or INN when judged by comparable numbers of total trainable parameters as shown in 3.2.

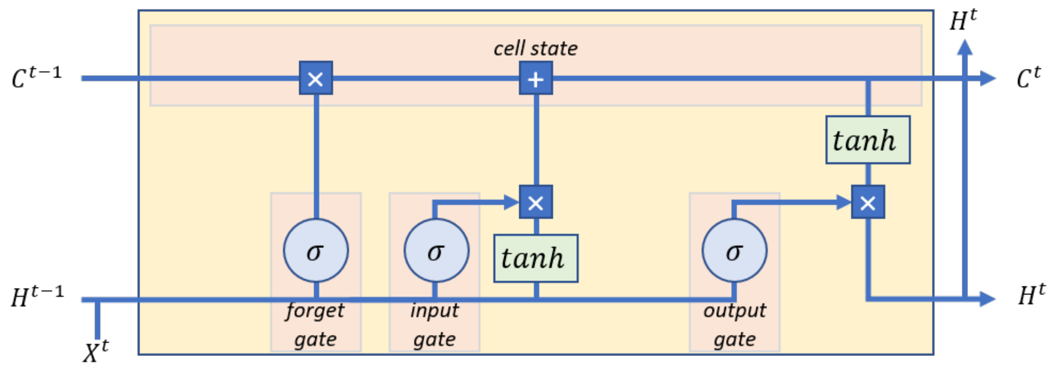


FIGURE 2.10: Gate structure of a long short-term memory (LSTM) network utilizing input gate (I^t), forget gate (F^t), cell gate (G^t) and output gate (O^t). Equations 2.16 demonstrate the Pytorch [24] implementation inspired by [31].

$$\begin{aligned}
I^t &= \sigma(W^{ii}X_t + B^{ii} + W^{hi}H^{t-1} + B^{hi}) \\
F^t &= \sigma(W^{if}X^t + B^{if} + W^{hf}H^{t-1} + Bhf) \\
G^t &= tanh(W^{ig}X^t + B^{ig} + W^{hg}H^{t-1} + B^{hg}) \\
O^t &= \sigma(W^{io}X^t + B^{io} + W^{ho}H^{t-1} + B^{ho}) \\
C^t &= F^t \odot C^{t-1} + I^t \odot G^t \\
H^t &= O^t \odot tanh(C^t)
\end{aligned} \tag{2.16}$$

For the purposes of benchmarking and establishing a comparative performance baseline on sequential data sets, LSTM models were used given their widespread popularity. Each LSTM model was tuned to approximate the number of trainable weights used within the iterative model architectures and both classes of models utilized identical learning rates and optimizers.

2.1.7 Multi-task learning

Multi-task learning (MTL) [7] was utilized during model training seeking to boost model stability and training speed. Multi-task learning is the practice of having a single model attempt multiple tasks during the training process using a shared data set [29]. The use of MTL also carries a regularization effect, often reducing the risk of overtraining through the use of shared representations across the weight space. The effect of this regularization often encourages models to train more quickly and with higher stability. The application of MTL can widely vary, but typically involves the use of multiple loss functions which inform gradient through the optimization function. Iterative neural network architectures may often serve as a natural use case for MTL, particularly when analyzing sequential data sets in which the interim steps in the sequence and corresponding data labels may vary from the final desired output. In the case of our anomaly detection time series, this occurs where each individual time step carries a ground truth binary output label indicating whether or not each element of the time series is an anomaly. At each final step in the sequence, the ground truth and resulting task of the model is to then predict the indexed location at which the anomaly within the series occurred. Attempting to predict the indexed location of anomalies at each sequential step would not be well suited for interim steps early in each sequence when it is likely the model has not even seen the anomaly yet. Instead it makes sense to consider multi-task learning in which we use two distinct loss functions. At each interim iteration of a sequence the model indicates whether or not each example it sees is the anomaly with a corresponding loss generated through binary cross entropy. On the final iteration of a sequence, the model predicts the indexed location of the anomaly and quantifies the loss using a standard cross entropy function. The resulting losses are then summed and propagated back through the optimization function at the end of each sequence. Following these principles MTL was utilized across both the RNN and LSTM networks during performance comparisons between model architectures.

2.2 Relevant metrics

2.2.1 Relevant performance metrics

Models were benchmarked against one another using both loss and accuracy as methods to compare performance. Aside from documenting model performance, loss functions served the purpose of providing input to the optimizer used in assigning model gradient as part of backpropagating errors. Since models were trained using multi-task learning, models also utilized multiple loss functions as discussed below. Additionally, each loss or accuracy metric was recorded both on a cumulative basis across each entire sequence and separately by isolating the output from the final step in each sequence. The details of each method are provided in the following section.

Loss

In quantifying model output error two distinct loss functions were utilized throughout the experiments. In all models at a minimum cross entropy loss was applied to the final categorical output. The cross entropy loss function L_{CE} is useful in cases where the ground truth is a multi-class nominal label with C classes where $\hat{y}$ is the model prediction and $y \in \mathbb{R}^{1\times C}$ corresponds to the ground truth or target value probabilities for each class label c as described in equation 2.17.

$$L_{CE}(\hat{y}, y) = -\sum_{c=1}^{C} y_k log(\hat{y}_k) \tag{2.17}$$

Additionally, in certain cases including some sequential data sets such as those used for anomaly detection, alternate loss functions were utilized such as binary cross entropy loss L_{BCE} as reflected in equation 2.18. In these cases interim ground truth values were used which are a simple binary [0,1] indicator at each step reflecting a two class outcome. Notably, the cross entropy loss shown in equation 2.17 is a generalization of binary cross entropy loss from equation 2.18 across multiple classes.

$$L_{BCE}(\hat{y}, y) = -y log(\hat{y}_k) + (1-y) log(1-\hat{y}) \tag{2.18}$$

Accuracy

Accuracy metrics were measured and aggregated using two approaches. The primary accuracy metric calculated and recorded across models focuses on the accuracy of the final iteration at the last step in each sequence, referred to as ACC_{end} and shown in equation 2.19. This metric aggregates the output y_i from only the final step or iteration within a sequence of length I among N examples. ACC_{end} relies on the boolean indicator function $1(y_{ni} = \hat{y}_{ni} \wedge i = I)$ which is positive only when the model output $\hat{y}$ matches the truth value y and $(\wedge)$ it is the final iteration i within a given example n. This measure of accuracy is better generalized across a variety of problems and model types while also providing clearer comparisons in sequences with unbalanced class distributions such as anomaly detection problems.

$$ACC_{end} = \frac{1}{N} \sum_{n=1}^{N} \sum_{i=1}^{I} 1(y_{ni} = \hat{y}_{ni} \wedge i = I) \tag{2.19}$$

An alternate approach used as a secondary metric across iterations and in sequential data sets aggregates full sequence accuracy as ACC_{seq}. This metric aggregates the output y_i from each interim step or iteration of a sequence of length I among N examples measured against the ground truth value $\hat{y}_i$ as shown in equation 2.20. ACC_{seq} uses the boolean indicator function $1(y_{ni} = \hat{y}_{ni})$ which is positive when the model output $\hat{y}$ and truth value y match for a given iteration i within a given example n.

$$ACC_{seq} = \frac{1}{NI} \sum_{n=1}^{N} \sum_{i=1}^{I} 1(y_{ni} = \hat{y}_{ni}) \tag{2.20}$$

2.3 Data sets

Several experiments were conducted seeking to examine model performance across a range of scenarios and conditions. Models faced problems that varied both in difficulty and format, with some receiving a single input example while others utilized series data. All data sets utilized in these experiments were derived from the underlying MNIST database and adapted as needed.

2.3.1 MNIST classification

The MNIST [12] (Modified National Institute of Standards and Technology) database is a common choice for machine learning exercises and features 70,000 labeled images often used for image recognition and classification problems. Among the combined total number of images 60,000 are categorized for use as training images while 10,000 are separately packaged for testing. Each image is a black and white square image with dimensions of 28×28 pixels that equate to 784 dimensions when presented as a flattened vector. The content of each image is a hand written single digit integer ranging from [0-9] and a ground truth label reflecting the integer that was depicted. Unmodified MNIST images can be seen in Figure 2.11. Throughout all experiments unless stated otherwise, the MNIST data set was divided into a training subset of 2000 images, a validation subset of 200 images and a testing subset of 200 images which were then adapted to meet the outlines of the problem accordingly and trained for a duration of 100 epochs.

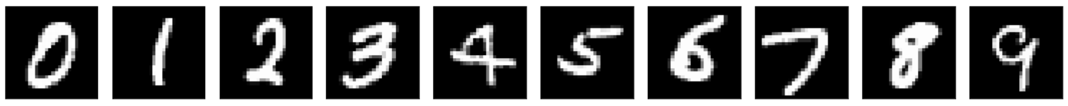
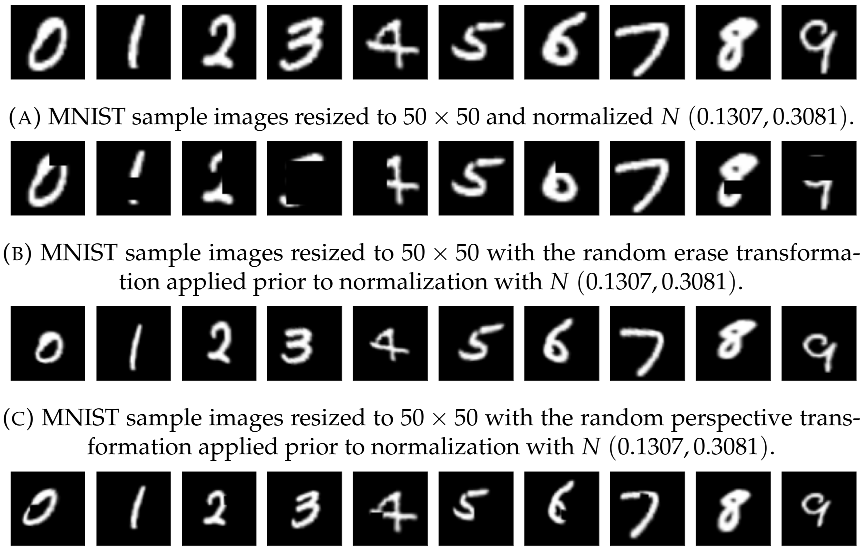


FIGURE 2.11: Examples of unmodified handwritten 28×28 pixel handwritten images from the MNIST database.

Classification transformations

Within early experiments MNIST was used towards traditional image classification tasks seeking to correctly identify the default labels associated with each image. However, several image transformations were applied with the objective of increasing the difficultly and creating more disparate comparisons among model structures. Since model implementations were performed through PyTorch using the PyTorch Lightning [10] wrapper, Torchvision [15] was a natural choice for transforms. As a first step, images were resized (torchvision.transforms.Resize) from 28×28 to 50×50, increasing the total number of input features in X to 2,500 from 784. Additionally, the features of each image are normalized (torchvision.transforms.Normalize) using the distribution defined as $N(0.1307, 0.3081)$ in adherence with guidelines from the torchvision documentation for the MNIST data set as shown in Figure 2.12a.


(A) MNIST sample images resized to 50×50 and normalized $N(0.1307, 0.3081)$.

(B) MNIST sample images resized to 50×50 with the random erase transformation applied prior to normalization with $N(0.1307, 0.3081)$.

(C) MNIST sample images resized to 50×50 with the random perspective transformation applied prior to normalization with $N(0.1307, 0.3081)$.

(D) MNIST sample images resized to 50×50 with both the random erase and random perspective transformations applied prior to normalization with $N(0.1307, 0.3081)$.

FIGURE 2.12: The examples above demonstrate the cumulative effects of transformations applied to samples from the MNIST data set.

Once the image has been resized each image is further transformed by randomly erasing (torchvision.transforms.RandomErasing) portions of the images prior to being normalized. The resulting image adds further difficulty to the problem as shown in Figure 2.12b. For this transform, probability is set to 1 while the range indicating the proportion of the image to be erased is specified using as a range of (0.02, 0.05).

As a second additional step prior to normalization, the random perspective transformation (torchvision.transforms.RandomPerspective) is applied to the image with probability of 1, using the default settings for distortion scale and fill values of 0.5 and 0, respectively. In combination both transforms increase the difficulty with examples shown in Figure 2.12c. The combined augmentations 2.12d of the MNIST data set create a more challenging problem for basic image recognition benchmarking among the models in this study.

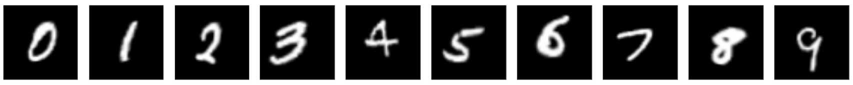
2.3.2 Sequential data sets

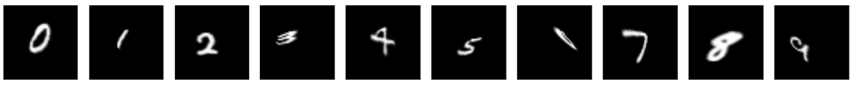
Two sequential data sets were generated for further analysis and tested, referred to as uniform anomaly detection and randomized anomaly detection. In each case, the task of each sequence is to select which image is unlike the others and provide its indexed location.

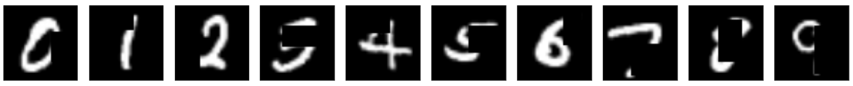
Sequential transformations

In the case of anomaly detection sequences in which transforms may be used to generate anomalies, the effects applied to the image are varied among several pre-defined combinations of transformations. In these cases, it is somewhat counter intuitive in that anomalies with less severe transformations are typically harder to identify than anomalies with greater applied distortions. This inverse relationship between severity of transformations and problem difficulty runs contrary to typical classification problems in which applying stronger transforms often raises the problem difficulty. With this in mind, the transformations used in anomaly detection sequences were generally softened with the objective of creating a problem set that would pose a sufficient challenge to differentiate performance across several architectures. The combinations of transformations which were utilized in these sequential anomaly detection problems are outlined below.

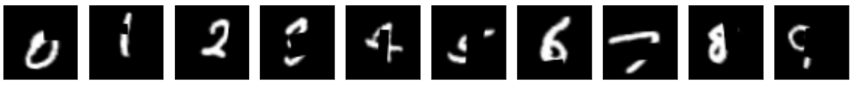
(A) **Baseline:** MNIST sample images resized to 50×50 and normalized $N(0.1307, 0.3081)$.

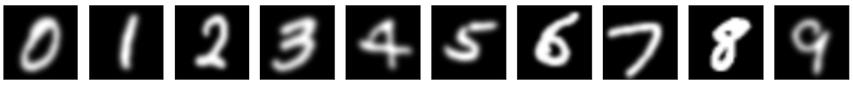

(B) **Perspective Small:** Random perspective applied as an interim step within the baseline transformation and setting distortion scale to default of 0.5.


(C) **Perspective:** Random perspective applied as an interim step within the baseline transformation, distortion scale increased to 0.9.


(D) **Erase:** Random erase applied prior to normalization within the baseline transformation using a range with proportion of erased area specified between 0.1 and 0.2.

(E) **Noise:** Gaussian noise transformation applied as an interim step with standard deviation of 0.01 to the baseline transformation.


(F) **Erasing & Perspective:** Both erasing and random perspective transformations applied sequentially as interim steps within the baseline transformation.


(G) **Blur:** Custom Gaussian blur transformation applied with kernel size 15 to the baseline transformation as an inner step in the transformation sequence.

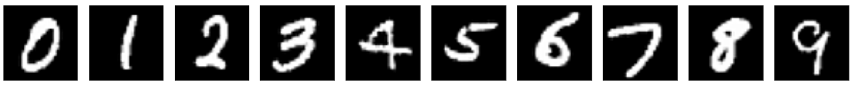

(H) **Sharpness:** Randomly adjusted sharpness with a factor of 4 applied within the baseline transformation.

FIGURE 2.13: The samples above from the MNIST data set are used to demonstrate the various combinations of transformations used in sequential data sets for anomaly detection. One of the above transformations is randomly selected and applied to a series of data while another randomly selected transformation is applied to a randomly assigned outlier in the sequence.

Baseline

- Resized $X \in \mathbb{R}^{50\times 50}$
- Normalized distribution $X \sim \mathcal{N}(0.1307, 0.3081)$

Perspective Small

- Resized $X \in \mathbb{R}^{50\times50}$
- Random perspective with default distortion scale of 0.5
- Normalized distribution $X \sim \mathcal{N}(0.1307, 0.3081)$

Perspective

- Resized $X \in \mathbb{R}^{50\times50}$
- Random perspective with default distortion scale increased to 0.9
- Normalized distribution $X \sim \mathcal{N}(0.1307, 0.3081)$

Erase

- Resized $X \in \mathbb{R}^{50\times50}$
- Random erasing with proportion of erased area to range between (0.1, 0.2)
- Normalized distribution $X \sim \mathcal{N}(0.1307, 0.3081)$

Noise

- Resized $X \in \mathbb{R}^{50\times50}$
- Gaussian noise applied with a standard deviation of 0.01
- Normalized distribution $X \sim \mathcal{N}(0.1307, 0.3081)$

Erasing & Perspective

- Resized $X \in \mathbb{R}^{50\times50}$
- Random erasing with proportion of erased area to range between (0.1, 0.2)
- Random perspective with default distortion scale increased to 0.9
- Normalized distribution $X \sim \mathcal{N}(0.1307, 0.3081)$

Blur

- Resized $X \in \mathbb{R}^{50\times50}$
- Gaussian blur applied with kernel size 15
- Normalized distribution $X \sim \mathcal{N}(0.1307, 0.3081)$

Sharpness

- Resized $X \in \mathbb{R}^{50\times50}$
- Randomly adjusted sharpness with a factor of 4
- Normalized distribution $X \sim \mathcal{N}(0.1307, 0.3081)$

Uniform anomaly detection

The uniform anomaly detection data set presents the simpler of the two anomaly detection tasks. In this case an MNIST image is selected and two instances are generated using unique combinations of transformations from the list illustrated in Figure 2.13. A sequence is then created using identical copies of one of the transformed images while the other then replaces one random instance in the sequence to create the "anomaly". In this case, all of the images in the sequence except one are identical copies with the one exception being the same image but with a different combination of randomly applied transformations. A final blank image appended and labeled with the ground truth value of the index location of the anomaly, triggering the algorithm to generate an estimate. To a degree this task embodies a memory problem as the variable for sequence length is increased. A series of examples of the uniform anomaly detection data set is demonstrated below with sequence length 9.

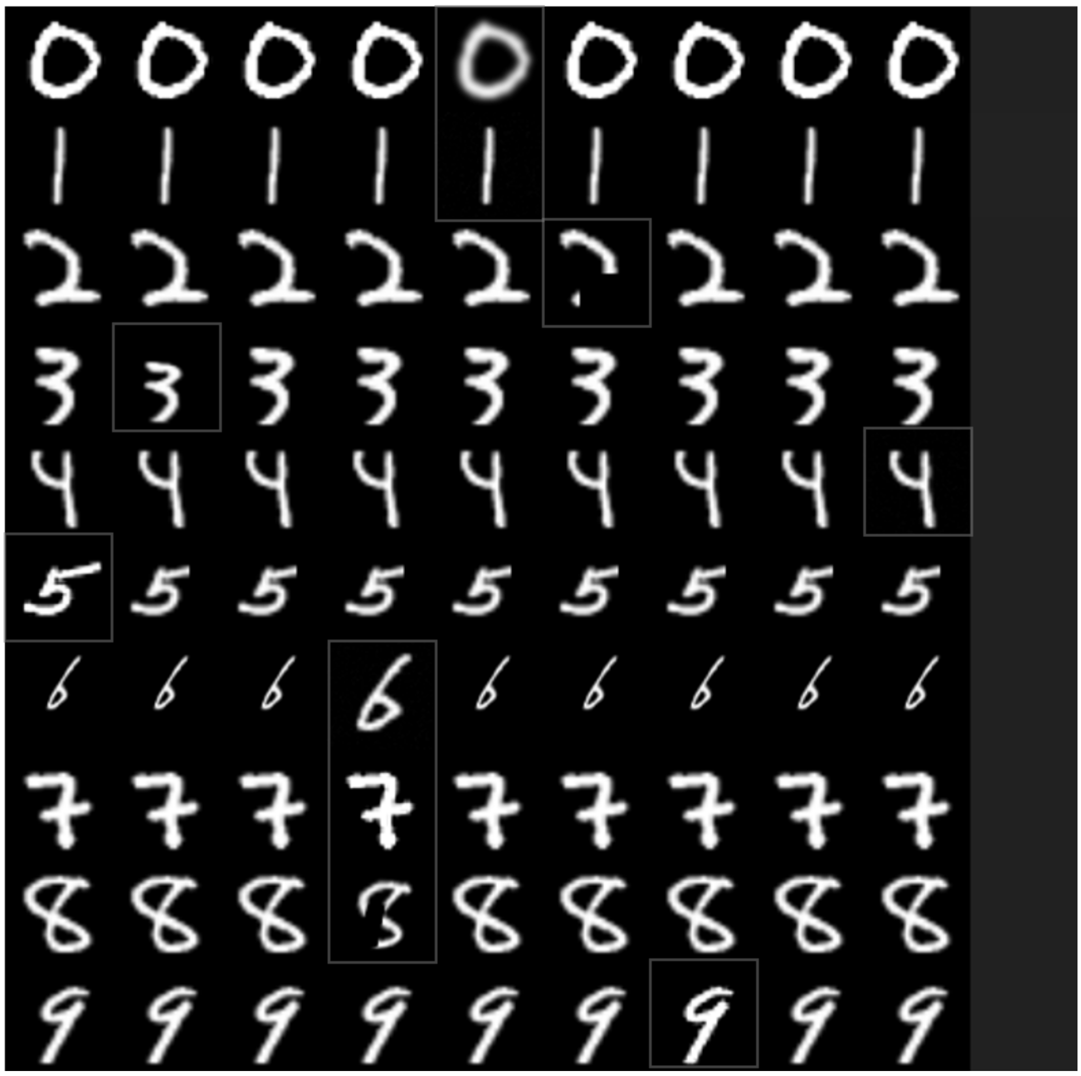


FIGURE 2.14: Examples of multiple series from the uniform anomaly detection data set with sequence length set to nine. Each series is presented horizontally with the anomaly outlined by a light grey square.

Random anomaly detection

The random anomaly detection data set represents a much more challenging problem in which a series of N different images from MNIST reflecting the same ground truth label are selected. All of the images will have the same randomly selected combination of random transformations applied while the other will have a different randomly selected combination of random transformations applied. In this case for example, N-1 different examples of a handwritten number X are selected from the MNIST data set and given the same randomly selected combination of random transformations while another different example of handwritten number X is given a different randomly selected combination of random transformations. Aside each copy of the images being different at the onset, each of the transformations may be applied differently. For instance, each randomly cropped, oriented or shifted image would have the transformation applied differently in a randomized way, yet would not be considered anomalous while the other randomly selected example with its own set of transforms would be the anomaly. A series of examples of the random anomaly detection data set with sequence length 9 is provided below. As before, the last blank image is paired with the ground truth index location of the anomaly and indicates the algorithm should provide its estimate. Further increasing the length of this data set appears to increase the challenge in terms of memory, problem complexity and by reducing certainty by way of more competing options.

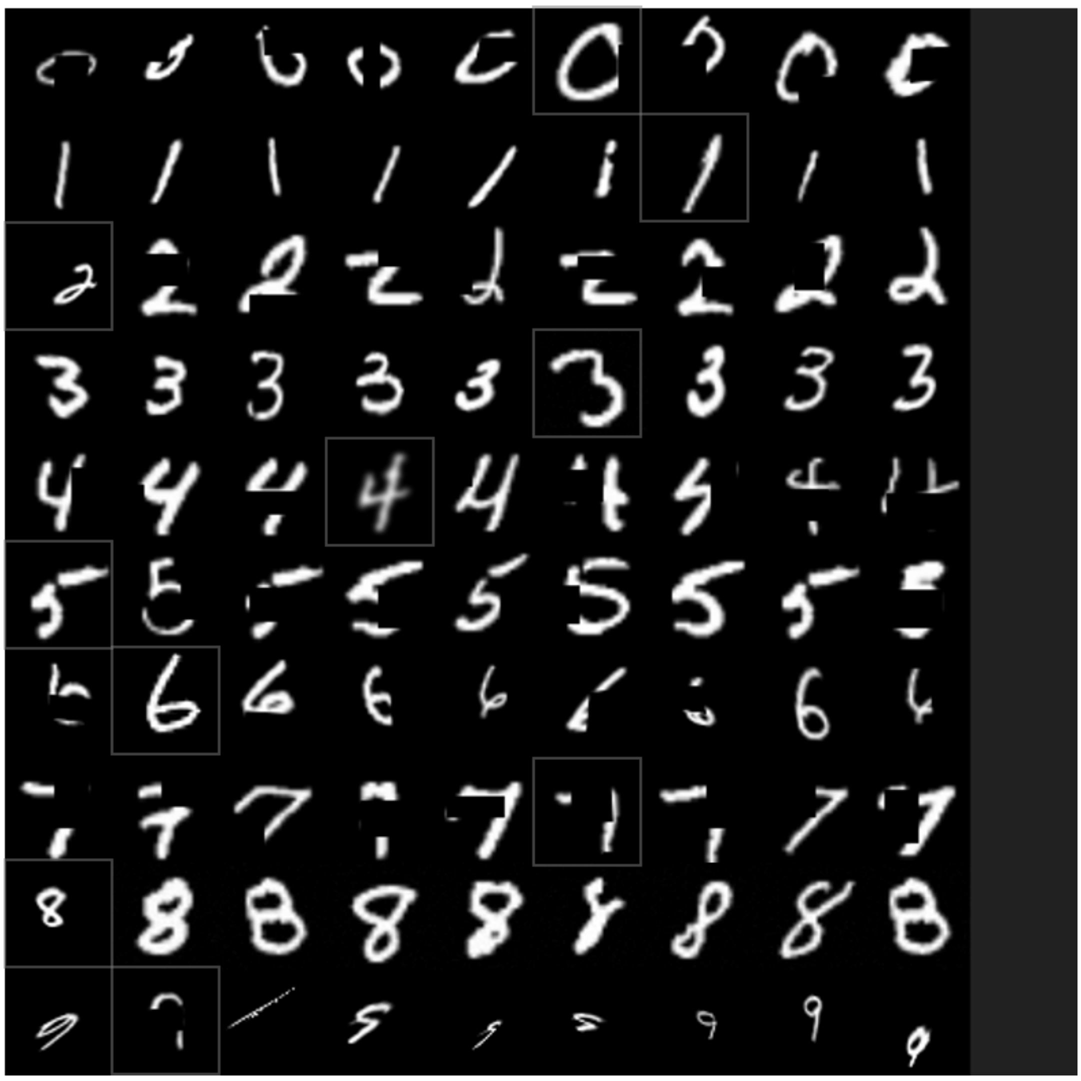


FIGURE 2.15: Examples of multiple series from the random anomaly detection data set with sequence length set to nine. Each series is presented horizontally with the anomaly outlined by a light grey square.

Chapter 3

Results

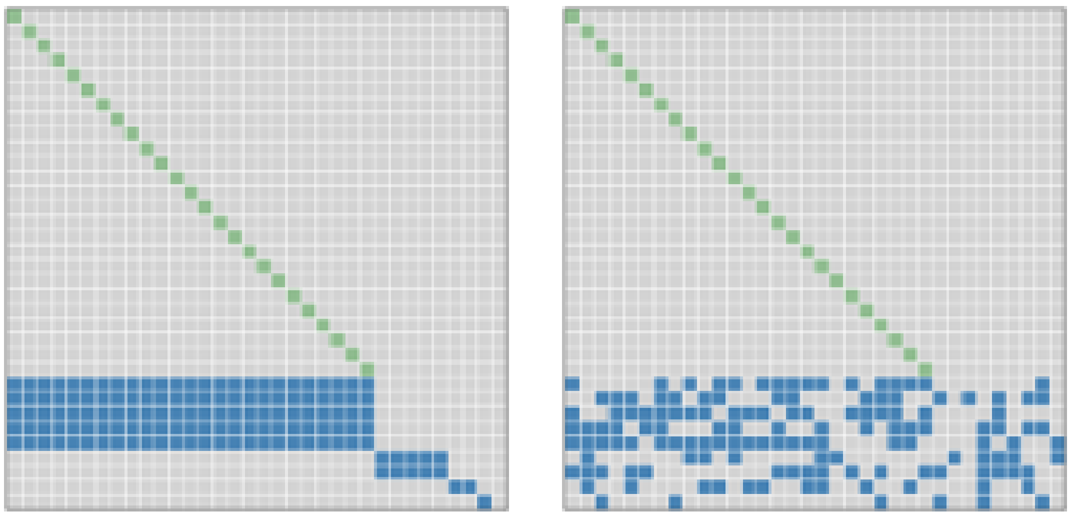
3.1 Overview

Experiments were conducted across several areas of focus. Initial efforts using the iterative network architecture were focused on determining the guiding principles associated with the location and arrangement of trainable parameters within the weight space as detailed in Section 3.2. Additionally, general comparisons were conducted in which INN performance was benchmarked against LSTM models with comparable numbers of trainable weights using the sequential anomaly detection problems in Section 3.3. Incorporating both sets of findings, further rounds of experiments were conducted exploring the use and benefits of sparsely arranged parameters within the weight space. The concept of sparsely parameterized neural networks is explored from several angles in Section 3.4. Initially, sparsity is examined in instances with number of trainable parameters left to vary while weight matrix dimensions remain fixed 3.4.1. Then, the effects of sparsity are explored with the number of trainable parameters fixed and matrix dimensions left to vary 3.4.2. All experiments were conducted by implementing neural networks in PyTorch [24] Lightning [10]. Training was conducted across both Colab Pro and Turing High Performance Computing at WPI while results were tracked through Weights and Biases [4].

3.2 Relative importance of model structure

Conventional wisdom has presented the layered arrangement and dimensions of parameter matrices as important hyperparameters relevant to model performance. A central feature of the INN architecture is the implied weight space initialized as untrainable zero matrices which surround the layers within a feed forward network. A natural line of reasoning would be to question whether these neatly organized dense matrices serendipitously represent the optimal network structure among the available weight space, a scenario that would appear unlikely. In the interest of testing the influence of weight placement on model performance, initial experiments were conducted exploring the impact of randomly varying network structures on training and test set performance. These experiments were designed by subdividing weight matrices into small clusters of weights which could then be randomly reassigned within the weight space. In a given test run, placement of trainable weights would be randomly varied in a manner in which the aggregate number of trainable weights within the model would remain largely unchanged, yet the overall architecture of the model would be modified for "all-else-equal" comparisons among randomly varied model structures. The resulting findings ran contrary to general expectations with little demonstrated differentiation between structured layers and random weight assignments as follows.

For each run, a base network was constructed with characteristics identical to those of a four-layer feed forward MLP with layers starting from an input dimension of 2,500 and with layer output dimensions of 500, 200, 100 and 10 sequentially as demonstrated by example Model 0 in Figure 3.1a. Each model used ReLu activation functions with a learning rate of 1e-3. In constructing the INN, the iterative parent matrix was decomposed into smaller component matrices of 100 $\times$ 100 to establish a greater degree of granular control over network structure. In the special case of rows or columns with dimensions beneath that size threshold, those component matrices would retain that smaller initial dimension. In this case, that would mean component matrices were sized 10 $\times$ 100 along the bottom row and 100 $\times$ 10 along the last column while the matrix in the bottom right corner remained a square matrix of dimension 10 $\times$ 10.


(A) Model 0: Example of initial experimental network structure.

(B) Model 1: Example of randomized network structure.

FIGURE 3.1: Side-by-side comparison of an MLP representation using an Iterative Neural Network (INN) versus a randomized trainable weight architecture. Green weights reflect a 2500 $\times$ 2500 untrainable identity matrix.

For this initial series of experiments a traditional MNIST classification experiment was conducted with both the random perspective and random erase transformations applied as in Figure 2.12d from Section 2.3.1. Once the network had been created, 10 training runs were performed on the data set using the traditional four layer MLP representation to establish a base comparison. Afterwards, 10 training runs were performed with each INN first instantiated following the basic four layer MLP structure, at which point each component matrix randomly exchanged initialization and trainability parameters with another component matrix with individual probability P=0.2 prior to training. This process was then repeated again for an additional 10 training runs using exchange probability P=0.5 illustrated by the example of Model 1 in Figure 3.1b, for a total of 30 combined training runs. A single constraint was placed on the exchange of parameter settings with the objective in retaining a general comparability among models. Attribute exchanges were restricted to occur solely between component matrices not located in a row horizontally aligned with the input component X in Figure 2.4 given no activation function is applied to those rows.

Following the process outlined above, experiments were run with results somewhat counter intuitive to prevailing theory regarding the principles and importance of network architecture. As an illustration of these results, the two very different networks shown in Figure 3.1a and Figure 3.1b were trained under the conditions described above, with the resulting training profile illustrated in Figure 3.2. Despite stark variation in architectures, both networks followed the same training profile with final testing accuracy between both models virtually indistinguishable at 92.7%. On the surface these results indicate other factors aside from recognized principles of structure govern overall model performance while demonstrating an unexpected robustness to alterations in model structure.

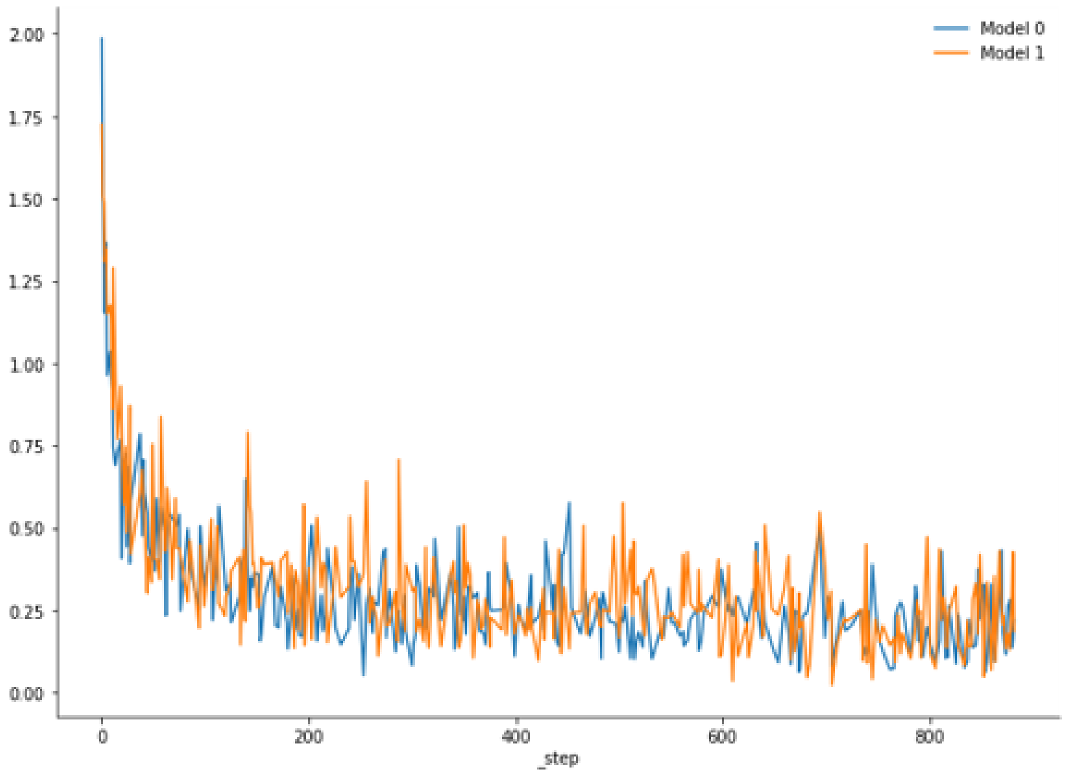


FIGURE 3.2: A comparison of training loss over 100 epochs between the layered architecture of Model 0 (3.1a) and the randomized architecture of Model 1 (3.1b) from Figure 3.1. The performance curves are indistinguishable despite the the dissimilar architectures.

Broadening the example to include results across all 30 runs arrives at an equally compelling illustration. By first plotting the 10 non-randomized training runs replicating a four layer MLP, we establish a benchmark performance set in Figure 3.3 with all models predictably showing similar training profiles.

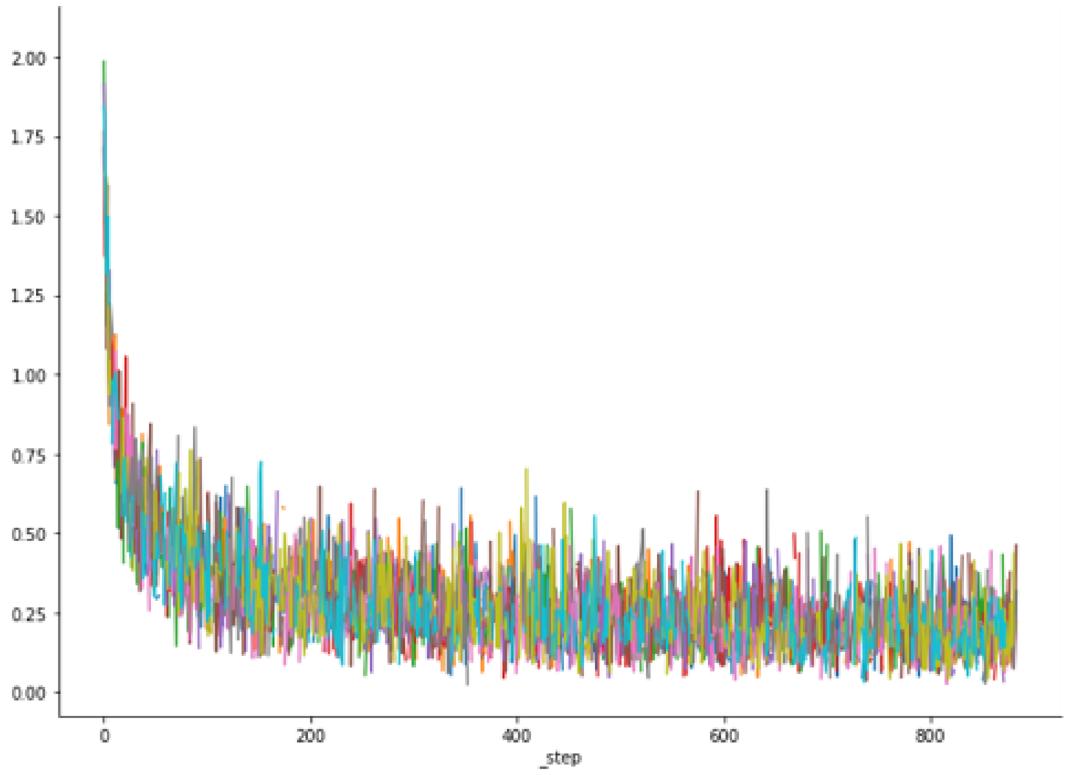


FIGURE 3.3: Performance curves demonstrating training loss over 100 epochs for 10 non-randomized models. These models reflect the layered architecture of Model 0 from Figure 3.1a, with repeated training runs establishing a baseline for broader comparison.

From the training runs with non-randomized structures in Figure 3.3, we now take the step of first plotting those training loss curves in blue before overlaying the training loss curves of models with randomized weight allocations in orange as shown in in Figure 3.4 and the results are compelling. Here 20 randomized runs are overlaid, essentially overwriting the other series of runs despite randomly assigned architectures. With little similarity in structure, the models train with indistinguishable performance. These results create compelling evidence that the concepts of layers and neatly defined weight matrices exist as arbitrary constructs. The principle of sequential model structure may well exist as a notational construct stemming from the use of nested functions as opposed to a governing characteristic of model performance.

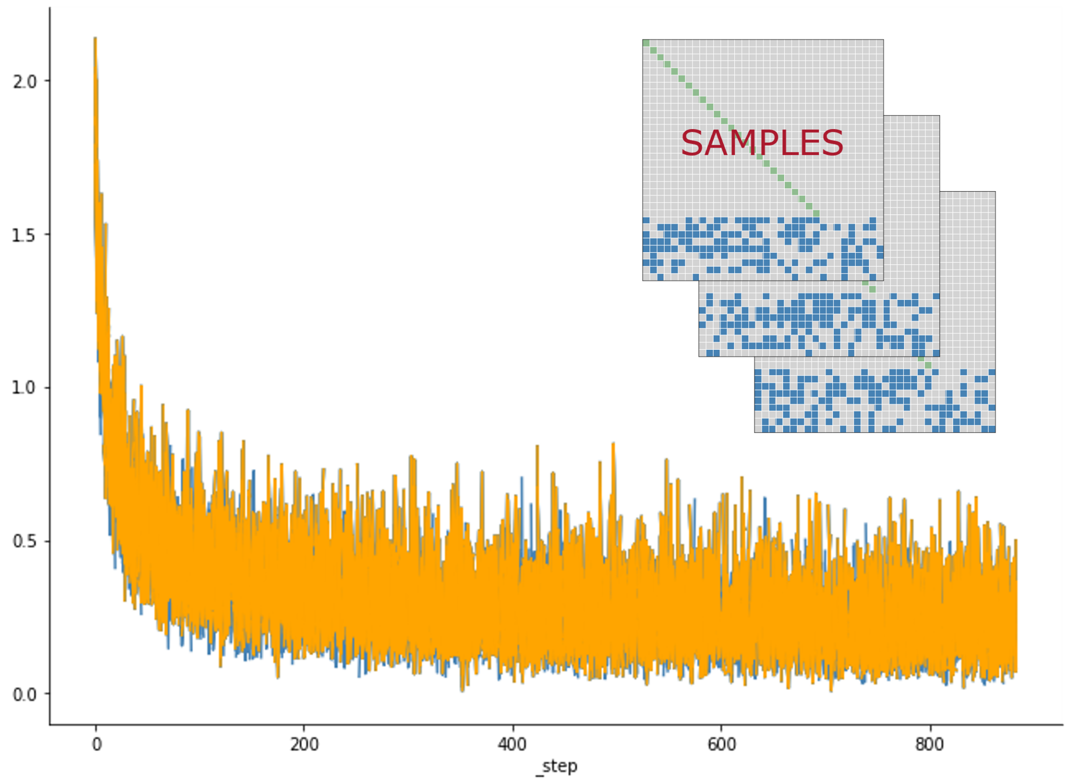


FIGURE 3.4: Training runs for models with randomized architectures reflective of Model 1 in Figure 3.1b are plotted over the course of 100 epochs in gold overlay. The results are plotted as an overlay against the training curves from Figure 3.3 which featured layered architectures. The output demonstrates the lack of differentiation between non-randomized and randomized model architectures.

The results held across transformations and generalized to testing with few exceptions. In each of these exceptions, the effects of randomization were severe enough to reduce data flow and create outliers as demonstrated in Figure 3.5 below. In the highlighted example, the absence of trainable weights within the portion of the bottom layer receiving input from the hidden state disrupted model performance. The lack of weights linking gradient through the hidden state caused the loss of the substantial majority of the model's embedded information resulting in outlying poor performance against similar models. Effectively, while the model retains the nominal trainable parameter count, few parameters have retained a gradient pathway connected to the loss function so the effective number of trainable parameters has been substantially reduced and weakened the comparison. These outliers demonstrate that while some aspects of structure appear to be potentially less significant than anticipated, the fundamental rules of neural network design still hold importance.

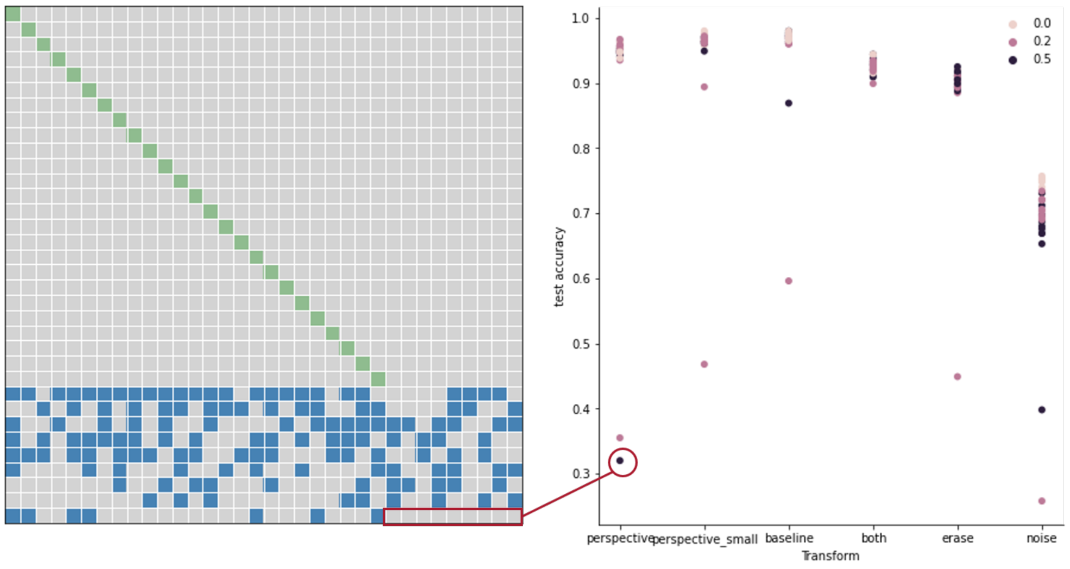


FIGURE 3.5: The rightmost plot demonstrates performance measured as end sequence test accuracy on the y-axis against input transformations of varying difficulty on the x-axis. Color coding reflects the amount of randomization applied to parameter architectures. With randomization applied, some outlier variation is introduced for small models. In these cases, randomization resulted in loss of gradient through the model and failure to train. Essentially, parameters were no longer located in positions to provide output from the weight space as shown on the left. This example makes explicit the intuitive point that in the extremes structure continues to matter.

The resulting implication and path forward for future research is best highlighted by combining these findings into several key points. The initial premise questioned the likelihood that current MLPs might serendipitously represent the optimal network structure among the broader potential weight space made explicit through the use of INN's. But this appeared unlikely. However, the initial results demonstrated little variation despite major structural changes, also raising the valid question of whether structure was ultimately irrelevant. Taken in the form of an absolute and explored using outlier cases, this seems demonstrably false and logically unlikely as well. The general path forward for further exploration questions whether between these two extremes, there are guiding rules to placement, initialization, and characterization of weights in iterative structures that are supportive of stronger performance. Specifically, if the arrangement of neatly demarcated layers is not a central feature of performance when constrained to comparable totals of trainable parameters, then the question becomes what factor or factors may be contributing to the performance variance as layers are altered. This question is explored further in Section 3.4 after benchmarking INN performance against an LSTM on sequential data sets in Section 3.3.

3.3 Model comparisons

Having examined the relationship between MLP and INN architectures on a standard image MNIST recognition task, a logical next step is to benchmark performance on sequential data sets against a common model architecture. To achieve this, INN's were deployed against both the uniform anomaly detection and random anomaly detection problems outlined in Sections 2.3.2 and 2.3.2. For these tasks, the iterative networks were instantiated with each individual trainable parameter in the weight space randomly assigned either an untrainable value of 0 or a random trainable float with hyperparameter probability R. The decision to retain the randomly assigned architecture within the INN's followed the prior findings from Section 3.2 which indicated no benefit from a more structured approach. Under these conditions, the INN by way of its generalized design assumes similar behavior to that of an RNN with trainable parameters randomly assigned throughout the weight space as discussed in Section 2.1.4. INN performance was comparatively benchmarked against an LSTM network as introduced in Section 2.1.6 with characteristics of each summarized in Table 3.1. For this series of experiments, the INN was specified in a manner that created a weight space proportioned like the weight space which could contain a four layer neural network with input dimension of 2500 and outputs for each sequential layer of 500, 200, 100 and 10. Within that weight space, parameters were individually specified as trainable and initialized with non-zero values based on random probability. The randomized parameterization of the iterative models resulted in small variations among total trainable parameter counts with the average being 1,343,631. In comparison, the LSTM network was structured with an input dimension of 2500, a hidden dimension of 130 and an output dimension of 10 resulting in a total trainable parameter count of 1,307,540. Both models employed multi-task learning with each using binary cross entropy for interim time steps indicating whether each time step represented an anomaly and cross entropy for the final model output indicating the index position of the anomaly within the sequence. Three runs were performed for each model with learning rates established at 1e-3.

Model	Weight Space	Avg Weights	Dim In×Out	Multi-Task	Learn Rate
INN	Sparse	$1,343,631$	2500×10	Yes	$1e-3$
LSTM	Dense	$1,307,540$	2500×10	Yes	$1e-3$

TABLE 3.1: Summary of model characteristics used to conduct initial performance comparisons between INN and LSTM models on anomaly detection tasks.

Against the uniform anomaly detection data set with sequence length of nine as described in Section 2.3.2 which poses the easier of the two sequential problems, the INN compared favorably against the LSTM. Performance was better for the INN as it significantly outperformed across all metrics, while validation end sequence accuracy significantly exceeding that of the LSTM through nearly all stages of training in Figure 3.6. Validation end sequence accuracy was introduced in equation 2.19 and aggregated over the validation data set following each epoch in the model training process. It would appear based on the strong INN performance against the LSTM that the inherent flexibility of the iterative model's architecture allows for a more effective allocation of weights.

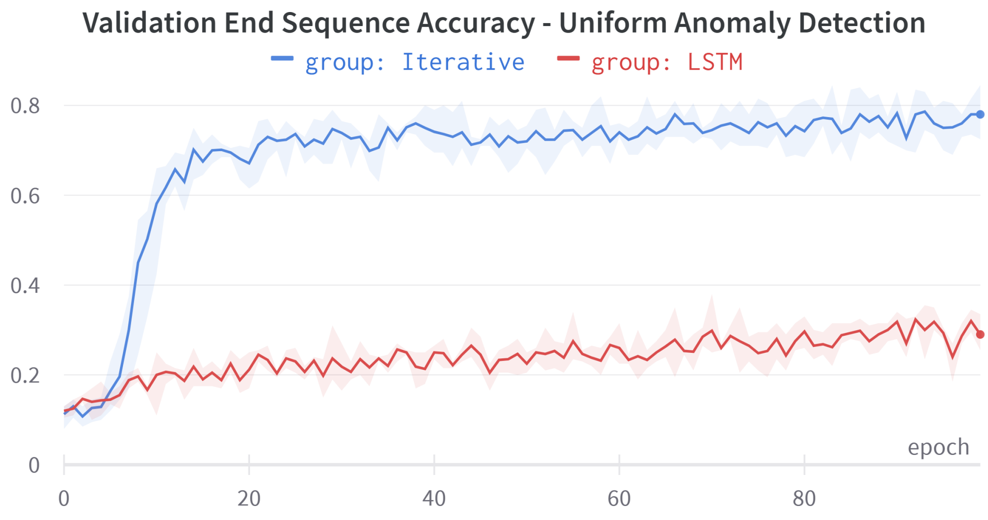


FIGURE 3.6: Performance comparison of Iterative Neural Network (INN) model in blue and LSTM in red with a comparable number of trainable weights. Average validation set accuracy tracked across 100 epochs for multiple runs. Performance reflects the uniform anomaly detection data set with sequence length 9. The INN is reflective of a randomized sparse RNN.

In extending sequence length to mid-length sequences, the comparison was repeated using the same underlying uniform anomaly detection data set. In this scenario the sequences were constructed with length of 19 with one such example given below in Figure 3.7. By extending the sequence length model performance was compared across settings requiring better retention of information. The use of longer sequences challenges the retained hidden state memory over a longer sequence while raising task complexity by increasing the number of comparisons and reducing the base accuracy associated with random guessing.

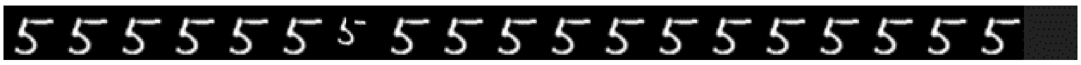
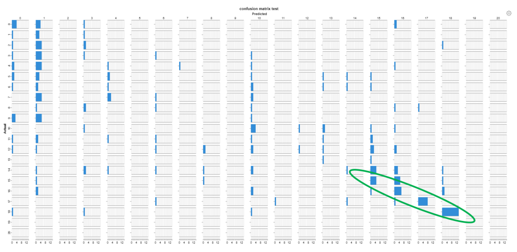
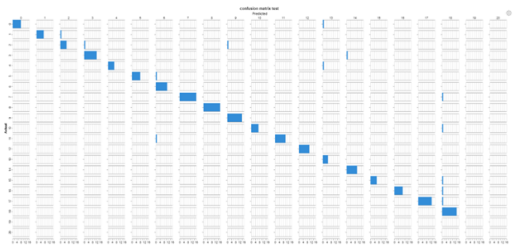


FIGURE 3.7: Uniform anomaly detection series of sequence length 19 with ground truth index value of 6.

A key advantage of the more generalized nature of the INN architecture lies in the model's inherent flexibility in determining the allocation of weights among various purposes during the training process. In comparison, the more structured arrangement of the LSTM may cause parameters to be distributed among competing objectives with both models constrained to similar approximate numbers of trainable parameters. As both difficulty and length of the anomaly detection challenges were varied, the flexibility of the INN appeared to serve as an advantage. The INN structure appears to contain more freedom to balance the need for embedded image detail against the need to recall that information later in the sequence. On the problems with sequence length 19 the performance remained bifurcated between the two model classes in favor of the iterative model. Additionally, it became clear when viewing a confusion matrix that the performance of the INN model was evenly distributed across anomalies occurring at all steps throughout the sequences. On the other hand, the LSTM models exhibited a clear recency bias, performing best on anomalies it had seen late in the sequence and thus more recently as highlighted in Figure 3.8.



(A) LSTM performance on uniform anomaly detection series with length 19.



(B) INN performance on uniform anomaly detection sequence of length 19.

FIGURE 3.8: Side-by-side comparison of confusion matrices contrasting performance of LSTM and Iterative Neural Network (INN) architectures on the uniform anomaly detection problem with sequence length 19. The LSTM network routinely demonstrated a clear recency bias, showing improved performance on anomalies that occurred later in the sequence.

Extending the analysis towards the more challenging task of random anomaly detection described in Section 2.3.2, the problem becomes more reminiscent of true anomaly detection. Against this challenge, the characteristic performance for each model architecture persisted with the randomized INN retaining a clear comparative advantage over the LSTM. In both cases the overall performance of both models decreased when measured by validation end sequence accuracy from equation 2.19, with the iterative model architectures seeing a larger decline. However, the iterative models continued to significantly outperform the LSTM based models at all stages of training and across all metrics. Figure 3.9 demonstrates the performance differential with respect to validation end sequence accuracy across epochs.

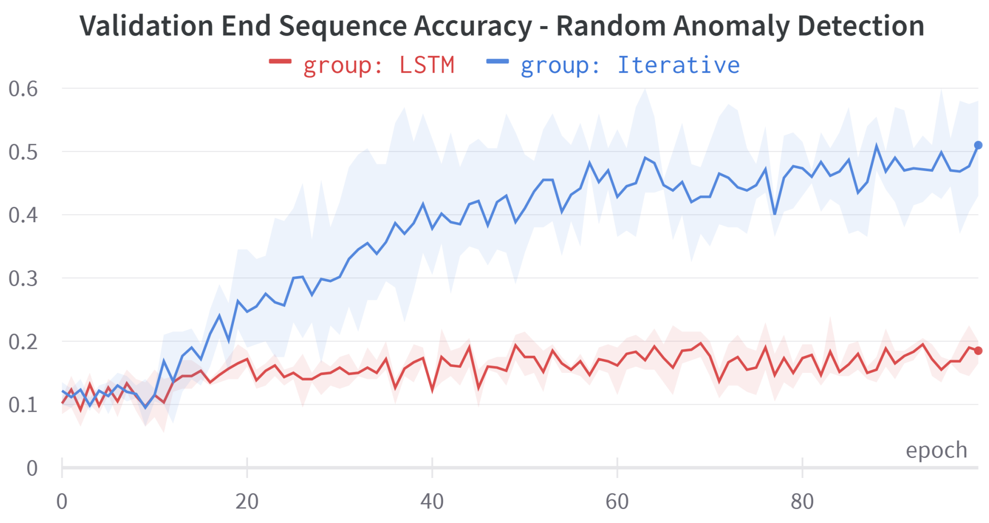


FIGURE 3.9: Performance comparison of Iterative Neural Network (INN) model in blue and LSTM in red with a comparable number of trainable weights. Average validation set accuracy tracked across 100 epochs for multiple runs. Performance reflects the random anomaly detection data set with sequence length 9. The Iterative Neural Network (INN) is reflective of a randomized sparse RNN.

As with the uniform anomaly detection data set, model performance tests were extended to cover medium length sequences with sequence length increased to 19 across the random anomaly detection data set with an example highlighted in Figure 3.10.

As a notable reminder, each image utilized in a random anomaly sequence is a different hand drawn version of the same number. In other words, a sequence of length 19 for the number six contains 19 different original handwritten sixes. Additionally, all but one of the images would have one common class of transformation applied as discussed in Section 2.3.2 with a separate class of transformation applied to the remaining image. However, even in cases where two numbers share the same type of applied transformation, each application of the transformation may differ and in a sense be unique unto itself as each contains its own randomization. For instance, if the eighteen non-anomalous sixes each had random crop applied, each of those unique sixes would in turn contain a unique application of the random crop transformation. This becomes particularly difficult in the case of random perspective since by nature of being hand written samples, each image is already randomly oriented with its own perspective prior to application of any transformation.

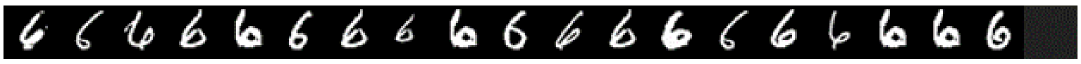
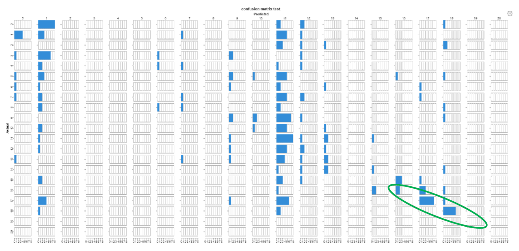


FIGURE 3.10: Random anomaly detection series of sequence length 19 with ground truth index value of seven.

Under the mid-length anomaly sequence the iterative model took more epochs to develop a strong advantage. However, the ultimate performance was very robust with validation end sequence accuracy eventually achieving levels on par with performance on the shorter length sequences and continuing to outperform LSTM's.

As with the earlier data set, the LSTM models exhibited a recency bias shown in Figure 3.11a with stronger performance on outlier examples that occurred later in the sequence. On instances where anomalies occurred early in the sequence the LSTM failed to improve beyond random guessing while the INN performed well in all cases as shown in Figure 3.11.


(A) LSTM performance on random anomaly detection series with length 19.

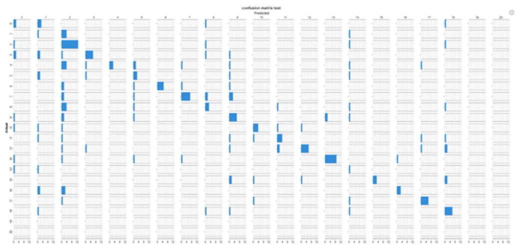

(B) INN performance on random anomaly detection sequence of length 19.

FIGURE 3.11: Side-by-side comparison of confusion matrices contrasting performance of LSTM and Iterative Neural Network (INN) architectures on the random anomaly detection problem with sequence length 19. The LSTM network routinely demonstrated a clear recency bias, performing better versus itself on anomalies that occurred later in the sequence.

3.4 Sparsity of the weight space

A natural area of interest following from the exploration of randomly arranged parameters within INN architectures lies with the influence of sparsity in determining model performance. Sparsity is interrelated with the aspect ratio between the depth and width of the available weight space and the corresponding number of trainable weights contained within that space. Here its worth mentioning that owing to the iterative nature of the INN architecture, the iterative matrix representing $f(X)$ in 2.4 is constrained to be square to ensure the ability for repeated iterations. This in turn causes the aspect ratio between the dimensions of the weight space to be governed by this constraint. As also show in Figure 2.5, the trainable weight space in these experiments is constrained to the area below the input matrix. This constraint exists because the rows of the matrix recursively feeding the input matrix are wrapped in an identity function as opposed to a nonlinearity. The result here is that each increase in the vertical depth of the weight space is accompanied by an equivalent increase in the associated horizontal width of the weight space, although the proportional impact is larger for the depth component as the weight space has a greater width than depth. This occurs due to the aforementioned absence of weights within the region horizontally associated with the input matrix. Often, the depth of the weight space is associated with a network's potential ability to model functional complexity. In Section 3.2 the findings demonstrated no discernible relationship between the ordered arrangement of network parameters in conventionally defined layers within a weight space versus randomized model architectures. In the act of creating these randomized arrangements of parameters, the weight space began to resemble a sparse matrix.

This section explores the relationship between the proportion of trainable weights within a given weight space and model performance under two scenarios. In the first case in Section 3.4.1, the dimensions of weight space are treated as a fixed constraint with the effect that varying model sparsity also varies the combined number of trainable parameters within the model. In the second case in Section 3.4.2, we explore the related concept of model sparsity in which the total number of trainable parameters is treated as a fixed constraint with the dimensions of the weight space left to vary. Using this arrangement, increasingly sparse weight arrangements also increase model depth. Both principles are explored in the following subsections. In the course of implementation, whether a specific parameter would be initialized as trainable was determined by the variable R. Within this context, a random number r generated from a uniform distribution for a given parameter within the range [0,1) would result in that parameter being trainable when $r < R$. Using a specific example, model sparsity of 0.1 would correspond to $R < 0.1$.

3.4.1 Sparsity of the weight space with fixed dimensions

We first explore the concept of sparse weight matrices in which the dimensions of the weight space are fixed and we vary the proportional presence of weights within that fixed space. Within this context, a sparsity of 0.02 would correspond to a 2% independent probability for each of the weights within the available weight space to be initialized as a non-zero trainable parameter and a sparsity of 1.0 would indicate a 100% probability of this occurring. Given the fixed dimensions of the weight space, this variable either directly reduces or increases the number of trainable weights distributed within the space. The results here are interesting with performance exhibiting low sensitivity to sizable variations in parameter sparsity.

This concept was initially tested using the the uniform anomaly detection data set with sequence length nine. For these experiments an INN was employed using a weight space in which setting the sparsity equal to 1.0 resulted in approximately 1.9 million trainable parameters with learning rates across all models set to 1e-4. Performance was benchmarked against a traditional LSTM network with approximately 1.3M trainable weights which would be roughly equivalent to a sparsity of 0.69 in the competing iterative network. Performance on the iterative networks began to deteriorate marginally as sparsity fell to 0.2, monotonically declining as sparsity was reduced to 0.1, 0.05 and ultimately 0.02. With sparsity set to 0.2, experiments demonstrated marginally delayed performance across the training curve in Figure 3.12 as measured using validation accuracy over epochs. As sparsity was further reduced to levels of 0.1 performance began to exhibit steeper declines along the training curve and ultimately never quite reached the levels achieved using higher sparsities. As sparsity was reduced further to 0.05 and 0.02, the erosion of performance became more pronounced across all points in the training curve.

However, benchmarking against the LSTM provided some interesting perspective to the iterative network performance. For all levels of sparsity, the iterative network was able to far exceed the final performance of the LSTM network across all metrics including validation end sequence accuracy. This includes the iterative network with sparsity of 0.02 which reliably reached average validation end sequence accuracy 2.19 levels of approximately 70% after roughly 70 epochs of training versus the LSTM which peaked below 50% on the same metric near the 90th epoch. This performance difference is compelling in light of the differential in trainable weights, with the 0.02 sparsity INN containing approximately 38,000 weights versus the roughly 1,307,000 weights contained in the LSTM. Additionally, for all sparsities above 0.05 the iterative network trained more rapidly than the LSTM across nearly all points of the training curve. Within this context it becomes clear the added architectural flexibility of the generalized network architecture may serve as a clear benefit within certain scenarios versus the LSTM when constrained by total number of trainable parameters. A compelling argument can be made that the iterative network may have been overparameterized for the simpler challenge of the uniform anomaly detection data set given the robustness against reduced parameterization. However the question also arises as to whether the decline in model performance at lower sparsity values is attributable to the effects of reduced parameterization coupled with the reduced interconnectedness between the weights themselves inherent in the less dense arrangement.

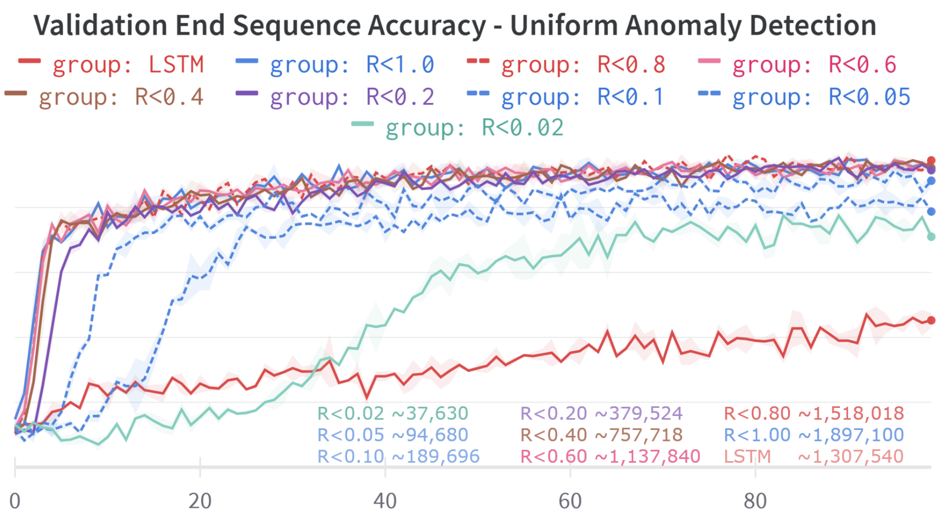


FIGURE 3.12: Various sparsity levels applied to the Iterative Neural Network (INN) weight space with fixed dimensions and total parameters left to vary. Performance is benchmarked against an LSTM model in solid red. The proportion of randomly distributed trainable weights in the weight space is depicted through the threshold variable *R*. Performance is measured with validation end sequence accuracy on the uniform anomaly detection data set of sequence length 9 over the course of 100 training epochs. The lower key indicates the number of weights corresponding to each level of iterative model sparsity.

The effects of sparsity on model performance were explored further using the randomized anomaly detection sequence, with sequence length again set to nine. The results remained consistent with this structure despite all models experiencing reduced accuracy throughout all stages of training given the more difficult task. In this case, end sequence validation accuracy peaked at a lower level of approximately 60% for fully trained iterative networks of varying sparsities. Contrary to initial expectations, by increasing the problem difficulty the effect of sparsity was less pronounced. Under these conditions, INN's of all sparsities aside from the 0.02 sparsity network (with validation accuracy peaking at 48%) ultimately reached similar levels of training performance over the 100-epoch training window. However, networks with sparsities less than or equal to 0.2 demonstrated progressively flatter training curves as shown in Figure 3.13. A performance gap of similar magnitude exists versus the LSTM network on this data set, with the LSTM peaking just above

30% validation end sequence accuracy near the 95th epoch but with results averaging closer to 25% over the majority of the training profile. Here the iterative network also demonstrates a clear performance advantage with nearly 20% stronger validation end sequence accuracy performance from the 0.02 sparsity INN featuring approximately 38,000 trainable weights versus the LSTM network with approximately 1,307,000 weights.

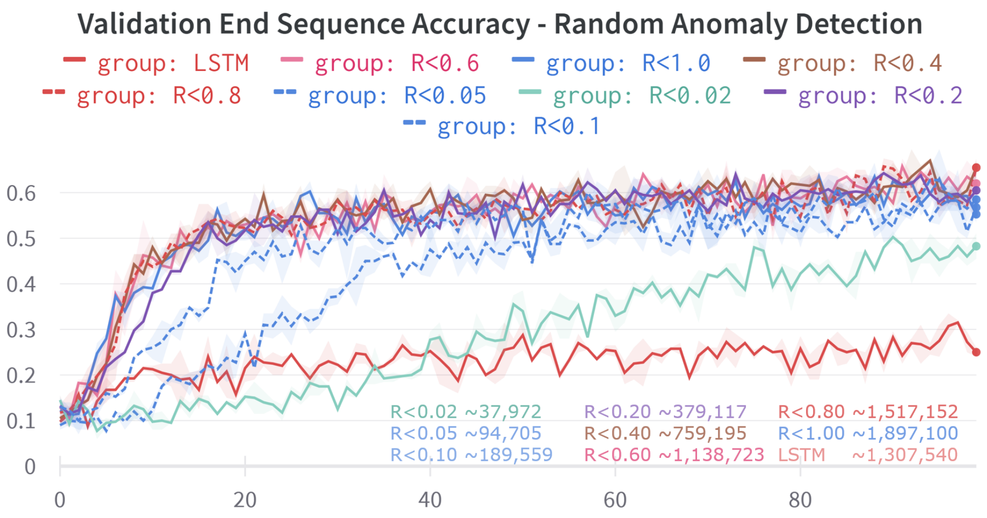


FIGURE 3.13: Various sparsity levels applied to the Iterative Neural Network (INN) weight space with fixed dimensions and total parameters left to vary. Performance benchmarked against an LSTM model in solid red. The proportion of randomly distributed trainable weights in the weight space is depicted through the threshold variable R. Performance is measured with validation end sequence accuracy on the random anomaly detection data set of sequence length 9 over the course of 100 training epochs. The lower key indicates the number of weights corresponding to each level of iterative model sparsity.

Against both data sets, the flexibility of the INN architecture outperformed the more rigid architecture of the LSTM by a substantial margin even in scenarios with far fewer parameters. Somewhat surprisingly, models in both cases demonstrated low sensitivity to large variations in parameter sparsity that altered trainable parameter counts beyond an order of magnitude. The advantage of iterative model architectures versus competing models suggests other proximal factors may play a significant role in model performance with one possibility being the depth or dimensionality of the weight space, explored in Section 3.4.2.

3.4.2 Sparsity of the weight space with fixed total parameters

Following the results from Section 3.4.1 which explore the role of parameter sparsity where the dimensions of the weight space remain fixed, experiments in this section were conducted adopting a slightly different approach. In this section, the influence of sparsity is examined where the number of total parameters is constrained as a fixed constant and the dimension of the weight space is permitted to vary. As explained in Section 3.4, the aspect ratio between the weight space dimensions is governed by the constraint that the iterative matrix be square. As the sparsity value decreases, the probability of any individual weight being trainable within the weight space decreases proportionally. As the total number of trainable parameters is a fixed constraint, this causes the dimensions of the weight space to expand. In other words, a network of sparsity 0.02 has approximately the same number of total trainable weights as a network of sparsity 1.0 albeit distributed differently among the changing dimensions of the weight space.

In the initial set of experiments outlined below, the total number of trainable weights was fixed to approximately 1.0 million for all models. Experiments were conducted using the random anomaly detection data set with sequence length of nine. The goal of experiments conducted in this regard is to explore whether the marginal performance decline previously witnessed with lower sparsity networks in Figure 3.13 is more closely aligned with the corresponding decline in trainable weight counts or the sparsity of the networks themselves. In these initial runs, training performance judged by validation end sequence accuracy across epochs carries low sensitivity to extreme variances in sparsity ranging from 0.02 to 1.0 as shown in Figure 3.14. As before, INN performance for all sparsity values comfortably outpaces LSTM networks with similar quantities of trainable weights. To a degree, the argument may again be made that the results reflect the potential overparameterization of the network versus the problem at hand, supporting additional training runs with fewer total trainable parameters below.

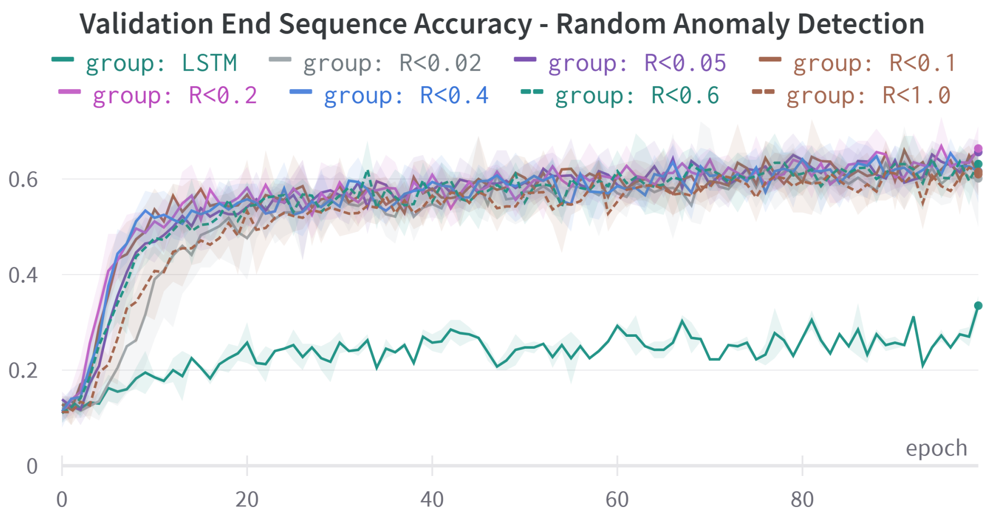


FIGURE 3.14: **Trainable weights fixed to approximately 1 million** for all models while various sparsity levels are applied to the Iterative Neural Network (INN) weight space. The LSTM network is shown for comparison in solid green. The proportion of randomly distributed trainable weights in the weight space is depicted through the threshold variable *R*. Performance is measured with validation end sequence accuracy on the random anomaly detection data set of sequence length 9 over the course of 100 training epochs. No clear impact is shown from altering sparsity levels alone within the INN architecture.

As a next step, the fixed weight count was reduced to approximately 0.1 million trainable weights for all models with results shown in Figure 3.15. Training runs were performed as before across sparsity values ranging from 0.01 to 1.0 with performance variations beginning to surface among the training curves. With the model now parameterized using an order of magnitude fewer trainable weights, performance began to deteriorate across both the most and least sparse networks. Under these tests, the more densely configured networks of sparsity 1.0 exhibited validation end sequence accuracy collapsing to a maximum of 20% at nearly 100 epochs versus other iterative networks reaching levels of 50-60% over much shorter time frames. Additionally, networks with sparsity values at either end of the potential range experienced marginal performance deterioration biased against higher sparsity values. In this regard, networks with sparsity values of 0.01 and 0.6 demonstrated marginally deteriorated performance early in training curves yet were ultimately able to perform on par with the other networks over the full 100 epochs of training. Results are strongest with little noticeable change in performance for networks with sparsity values in the range $[0.05, 0.2]$. Reducing the parameter account begins to highlight performance variance as it appears two effects are occurring at either extreme of sparsity values in the range $(0, 1]$. For sparsity values closer to the upper limit of 1, the dimensions of the weight space become increasingly compact with the lack in vertical depth specifically contributing to a loss of complexity in the representation of the solution space. In the specific case where sparsity is 1.0, performance has begun to deteriorate on par with the LSTM containing a similar number of weights. For sparsity values closer to 0, the network risks becoming underparameterized as the high degree of sparsity in the network begins to essentially disconnect some number of trainable weights from the pathway of gradient. The insight gleaned from these tests appears to reinforce the suggestion that while the number of total trainable parameters within a model is a relevant factor, beyond a sufficient baseline for the problem at hand, adding parameters has limited direct impact while risking overfitting models to the underlying data set. It appears that beyond the minimum required quantity of parameters, changes to the layered architecture of traditional feed forward networks may instead be inefficiently attempting to influence model performance through second order impacts on dimensionality of the weight space. Performance gains may be more efficiently realized by way of directly tuning model sparsity with improved training stability resulting in fewer trade offs.

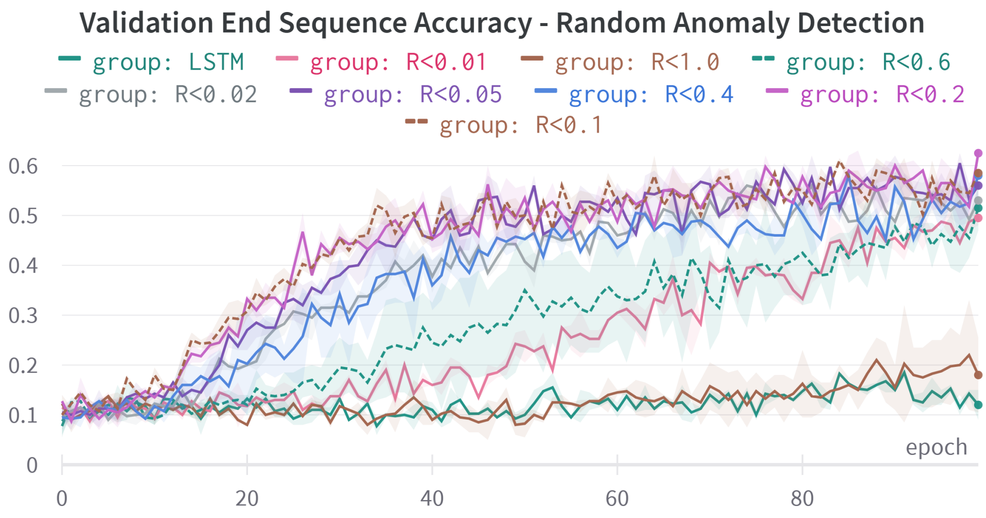


FIGURE 3.15: **Trainable weights are fixed to approximately 100,000** for all models while various sparsity levels ranging from 0.01 to 1.0 are applied to the Iterative Neural Network (INN) weight space. The LSTM network is shown for comparison in solid green. Performance is measured with validation end sequence accuracy over the course of 100 training epochs on the random anomaly detection data set of sequence length 9. With fewer trainable parameters, results begin to differentiate and are strongest for sparsity values ranging from 0.05 to 0.2.

To further explore this effect, the number of trainable weights serving as a fixed constraint was lowered to 50,000 weights for all models in Figure 3.16. Experimental runs were again performed for sparsity values ranging from 0.005 to 1.0 across the same data set. At this stage, performance began to more clearly separate with networks of sparsity 1.0 failing to learn entirely, which was reflected in validation end sequence accuracy of 10% approximating that of randomly guessing after 100 epoch. In networks parameterized with sparsity values closer to 1.0 and thus less weight space depth, the lack of vertical depth in the weight space appears to be forming a serious constraint on model performance. To a lesser degree networks with sparsity values of 0.6 experienced the same effect with validation end sequence accuracy lagging that of most networks with sparsity closer to 0.0 throughout training. After completing the full 100 epochs of training, validation end sequence accuracy for networks with 0.6 sparsity reached less than 40% in comparison to many networks averaging closer to 50%. At the other end of the spectrum extremely low sparsity values began to limit the flow of gradient, as networks with sparsity of 0.005 also showed very weak performance with validation end sequence accuracy tracking just above 20% after 100 epochs. Diminished performance in this regard was driven by the issue of weights becoming disconnected from the gradient path and resulting in an underparameterized network, accentuated by the low initial parameter count. The reduced number of parameters did see overall performance levels marginally decline to the area of 50% versus the approximately 60% levels demonstrated by the networks with trainable parameter counts of 0.1 million and 1.0 million over similar timelines. At 50,000 weights, the LSTM could no longer be defined for this task. However, in comparison performance was largely very strong for the INN models which continued to outpace the earlier LSTM containing 1.3M parameters from Figure 3.13. Best performance occurred for networks with sparsity values between 0.05 and 0.2 across the training curve.

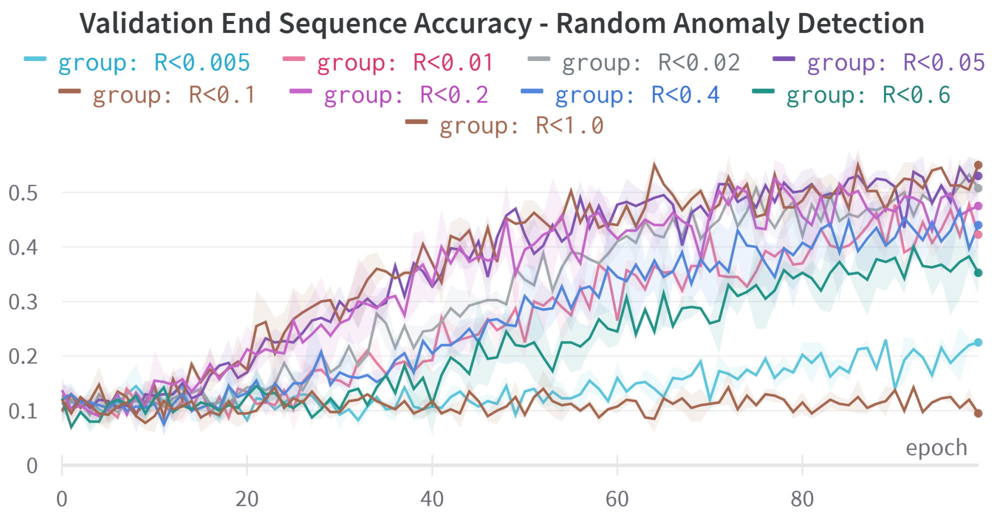


FIGURE 3.16: **Trainable weights are fixed at approximately 50,000** for all models while various sparsity levels ranging from 0.005 to 1.0 are applied to the Iterative Neural Network (INN) weight space. Performance is measured with validation end sequence accuracy on the random anomaly detection data set of sequence length 9 over the course of 100 training epochs. Under reduced levels of parameterization, the results become very differentiated with sparsity values between 0.05 and 0.2 performing best.

Taken a final step further toward the extremes, parameter counts were reduced to 10,000 for all models with sparsity values ranging from 0.005 to 0.2 as shown in Figure 3.17. At this level of parameterization, the networks were generally very underparameterized for the problem and the weight space had narrowed to a point at which it could no longer be defined for networks with sparsity values above 0.2 in addition to the LSTM network. Under these conditions, results for the remaining networks weakened and began to break down with the best achieving validation accuracy levels of 23% over 100 epochs. Worth noting, under these strained conditions the sparsity value of 0.1 appeared to perform reliably better than the others potentially indicating an approximate optimized level for network architectures in agreement with the results above.

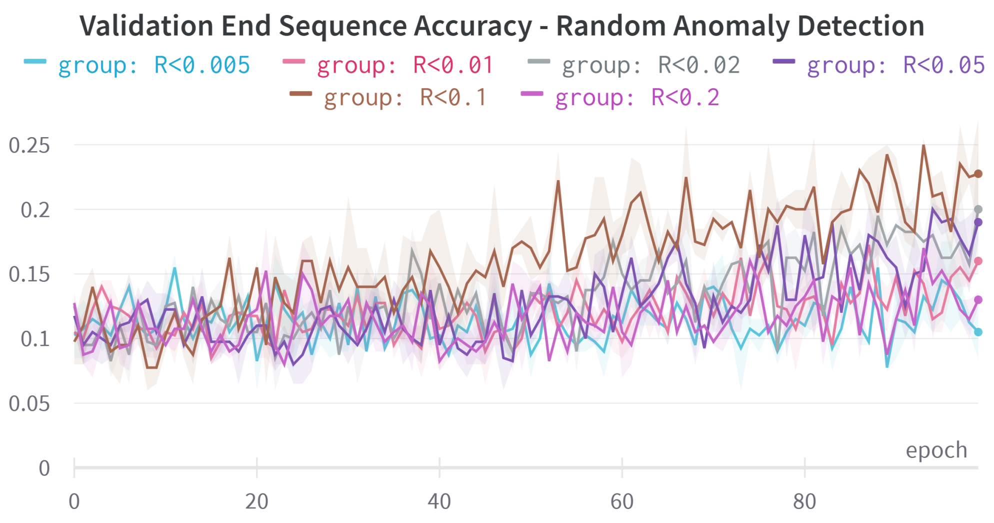


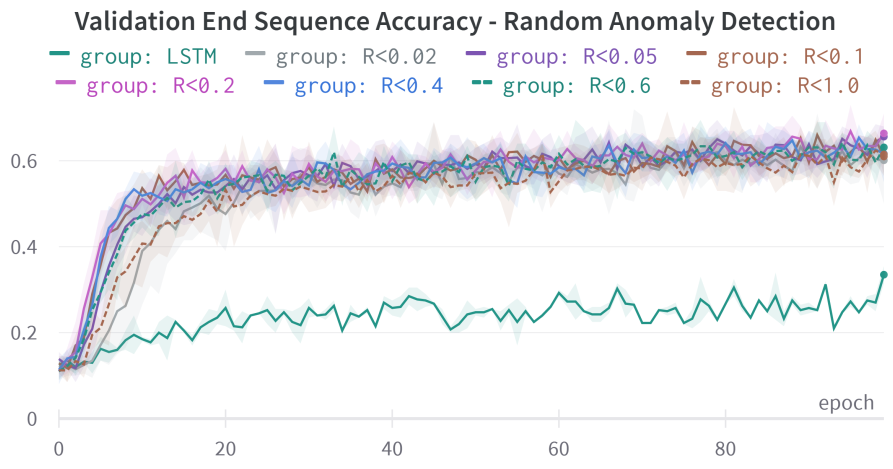
FIGURE 3.17: **Trainable weights are fixed to approximately 10,000** while various sparsity levels ranging from 0.005 to 0.2 are applied to the Iterative Neural Network (INN) weight space. Performance is measured with validation end sequence accuracy on the random anomaly detection data set of sequence length 9 over the course of 100 training epochs. Results are challenged by the effects of underparameterization with sparsity values between 0.05 and 0.2 performing best led by networks with a sparsity value of 0.1.

The clustering of results for most mid-range sparsity values in the experiments performed above point towards an optimal band for sparsity across models. Further, the results indicate the performance decline experienced for low sparsity values in Figure 3.13 were driven primarily by the associated decline in total trainable parameters. This perspective is reinforced by robust performance for adequately parameterized models at similarly low sparsity values and the overall deterioration in model performance even at higher sparsity values when total model parameterization was reduced to 50,000 and ultimately 10,000. As sparsity varied with dimensions of the weight space fixed, the resulting performance variations appear to be driven primarily by shifts in the total number of trainable parameters. As a result, model performance appears closely tied to ensuring depth of the weight space through sparsity once sufficient number of trainable parameters has been achieved. However, as with the experiments performed through the lens of sparsity with fixed dimensions of the weight space, results also remained generally robust to an array of hyperparameters. Specifically, as long as either sparsity remained within the central portion of the 0 to 1 range or the model remained heavily parameterized as with Figure 3.14, performance remained strong. In the latter case, large variations of sparsity showed little immediate impact.

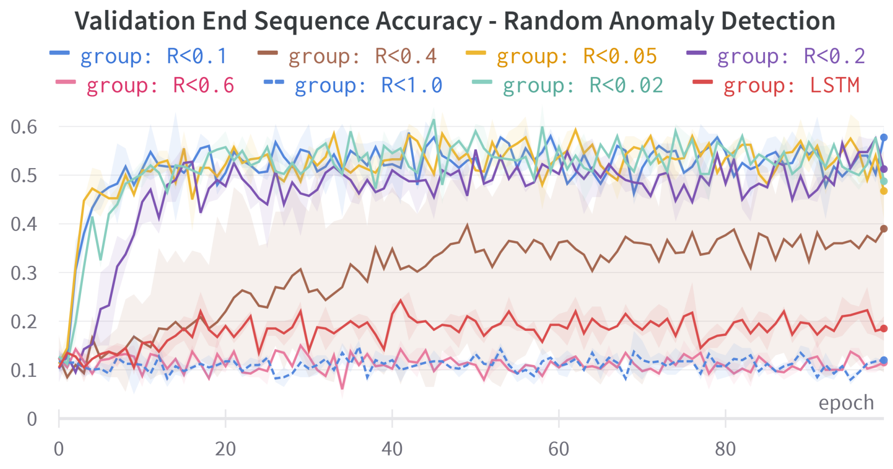
Taking into consideration the results from testing the impact of sparsity under varied conditions, a few things become clear. Network performance appears in this context to be primarily sensitive to achieving adequate vertical network depth from a sufficient base number of trainable weights as opposed to arbitrarily varying defined structures within the weight space. This can be seen as network performance was maintained across wide variations in number of trainable weights so long as adequate weight space depth is maintained through the use of sparsity. Moreover, even after significantly reducing the number of trainable weights and suboptimally reducing sparsity values, the vast majority of INN's appear to exhibit superior performance to popular structured architectures such as LSTM models on sequential problems. It appears the use of more optimal sparsity levels may create network structures less sensitive to specific hyperparameter tuning as performance remained generally robust across a wide array of defined weight spaces. To some degree the challenge often faced in model development of striking a proper balance between underparamaterized and overparamaterized networks appears to be accentuated by the widespread use of network structures with default sparsity values of 1.0. By using default sparsity values of 1.0 in models, practitioners may be creating a heightened sensitivity to parameterization through lack of sufficient depth in the weight space on the lower bound (likely taken for underparameterization), which must then be balanced against the risk of overparameterization on the upper bound. In the following section the research explores model training stability for networks incorporating sparsity.

3.4.3 Sparsity and model stability

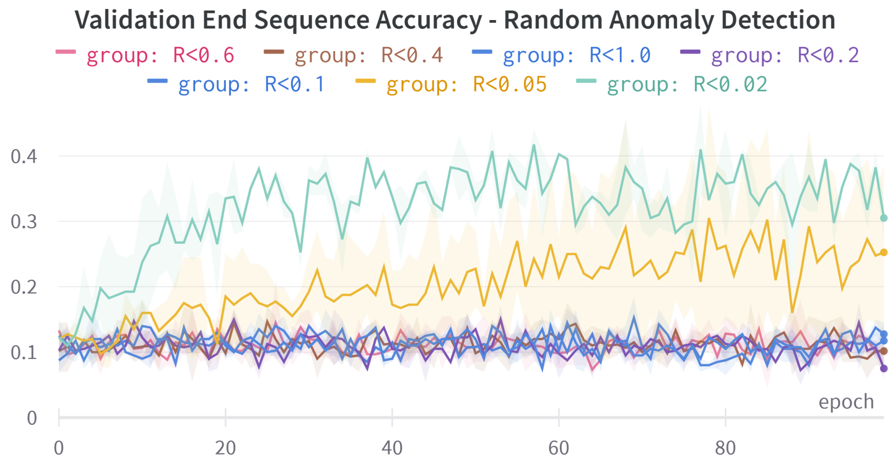
An additional observation surfaced regarding the effects of sparsity on model behavior during experimentation. As mentioned above, the use of sparsity significantly broadens the performance band for models by enabling more robust model performance across a wider array of hyperparameters. This performance stability can be observed as resilience against increases in the learning rate beyond levels that would often cause models to fail to converge. In the cases below, we begin with the performance graph shown earlier for a variety of sparsity values with the number of trainable weights fixed to 1.0 million and the learning rate set to 1e-4 on the random anomaly detection task. As before in Figure 3.14, the LSTM still markedly trails iterative model performance yet demonstrates some capacity to learn, while all INN's are demonstrating comparable performance curves plotted as validation end sequence accuracy over epochs.



(A) **Learning rate set at 1e-4**, INN performance is tightly clustered across sparsity values with the LSTM in solid green reflecting the stability of a low learning rate.



(B) **Learning rate increased by an order of magnitude to 1e-3** causing training stability to worsen. Performance begins to decouple with sparsity values closer to 1.0 failing to train.



(C) **Learning rate increased further to 4e-3**, performance breaks down across most networks with only sparsity values closer to 0.0 continuing to train.

FIGURE 3.18: With total number of trainable weights fixed to approximately 1 million, various sparsity levels are applied to the Iterative Neural Network (INN) weight space as the learning rate is markedly increased. Performance is measured by end sequence validation accuracy over the course of 100 epochs of training. Results are benchmarked using the random anomaly detection with sequence length 9.

However with all else constant, as the learning rate for each model is increased by an order of magnitude to 1e-3, performance becomes much more variable with many of the iterative models completely failing to train while LSTM performance erodes further. Notably, all sparsity values greater than 0.4 fail to show improvement over the training period with the sparsity value of 0.4 serving as the dividing line. Performance of models with 0.4 sparsity values became extremely variable with some models failing to learn and others eventually reaching performance near that of more sparse models at a more gradual pace and after an initial delay.

Pushing the learning rate higher yet to 4e-2 sees the effect become further pronounced as shown in Figure 3.18, with all models aside from INN's with sparsity values of 0.02 and 0.05 failing to train and producing end sequence validation accuracy of approximately 10%. In the case of the INN with sparsity set at 0.02, performance eroded but maintained consistent validation end sequence accuracy of approximately 35% through much of the training curve. Performance for the INN with a 0.05 sparsity value was volatile, with some runs failing to train while others achieved performance on par with the 0.02 sparsity value network.

These findings progressively illustrate the potential for improved performance stability realized by introducing sparsity into model architecture as stability improving with lower sparsity values. In the more extreme cases there appears to be no clear lower bound performance penalty as loss of gradient to parameters only serves to improve stability. However, under more typical conditions there clearly exists a tradeoff between reducing sparsity values to the point where the total effective parameter account is negatively impacted and the benefits to stability and model depth which must be considered. The optimization of these concerns potentially through the use of percolation theory presents an opportunity for future research.

3.4.4 Sparsity, aspect ratio and measuring task difficulty

The INN architecture is well suited for the alternate purpose of measuring task difficulty through a comparable quantitative metric across multiple domains and problem types. The INN architecture's ability to be applied across both sequential and non-sequential data sets provides versatility towards this task. Additionally, the randomized paramaterization is independent of specified constructs such as layers creating a problem agnostic architecture capable of scaling with problem difficulty. The quantified scaling of the INN to meet specific problem thresholds may in turn serve as a measure of problem difficulty.

Previously, Section 3.4.1 explored the concept of sparsity in which parameter matrix dimensions were fixed while the number of trainable weights varied with matrix sparsity. In the following Section 3.4.2, the number of trainable weights were fixed while matrix dimensions were left to vary with model sparsity. In this section exploring measurement of problem difficulty, sparsity itself becomes the fixed parameter while both number of trainable weights and dimension of the weight space are left to vary. Importantly, just as before the iterative matrix $f(X)$ faces a constraint of squareness which in turn governs the aspect ratio of the weight space as discussed in Section 3.4. The formulas governing the dimension of the weight space are solved using the quadratic formula and provided in equations 3.1.

$$\begin{aligned} R &= \frac{W^{train}}{W^{depth} \times W^{width}} \\ W^{width} &= \frac{(In + \sqrt{I^2 + 4W^{train}/R}}{2} \\ W^{depth} &= W^{width} - In \end{aligned} \tag{3.1}$$

Where:

$R =$ sparsity,
$In =$ number of input features,
$W^{train} =$ total trainable weights,
$W^{width} =$ width of weight space,
$W^{depth} =$ depth of weight space

To achieve this objective, a performance level and metric must first be specified for the given problem. Sparsity is then constrained to a fixed quantity. In this example sparsity is fixed at 0.2 while the other variables including number of trainable parameters and dimension of the weight space are left to vary. The model is initialized at a given number of trainable parameters and increased over multiple training instances until the performance threshold is met with the final number of parameters representing the difficulty of the problem as specified.

Problem difficulty may be considered in several ways. Firstly, the difference in weights required to achieve varying levels of performance on the same task can be used to gauge the relative difficulty of each performance level on that same task. Secondly, a given performance level on one task may be directly compared to a given performance level on another task using the same approach. In Figure 3.19, these methods are illustrated using random anomaly detection and uniform anomaly detection data sets with sequence length 9 as well as the basic MNIST classification task using the "both" transformation from Section 3.2. Using the least parameterized INN that can reliably converge for a given performance level, tasks are compared by number of weights required. The MNIST classification and uniform anomaly detection tasks are able to attain 80% end sequence accuracy over 100 epochs at comparable minimum parameterizations of 50,000 weights reflecting equivalent levels of difficulty between the two tasks. The random anomaly detection task converges towards a 57% validation end sequence accuracy at 60,000 weights. In comparison, this performance can be achieved on the uniform anomaly detection at 15,000 weights which quantitatively underscores the significantly more challenging nature of the random anomaly detection task.

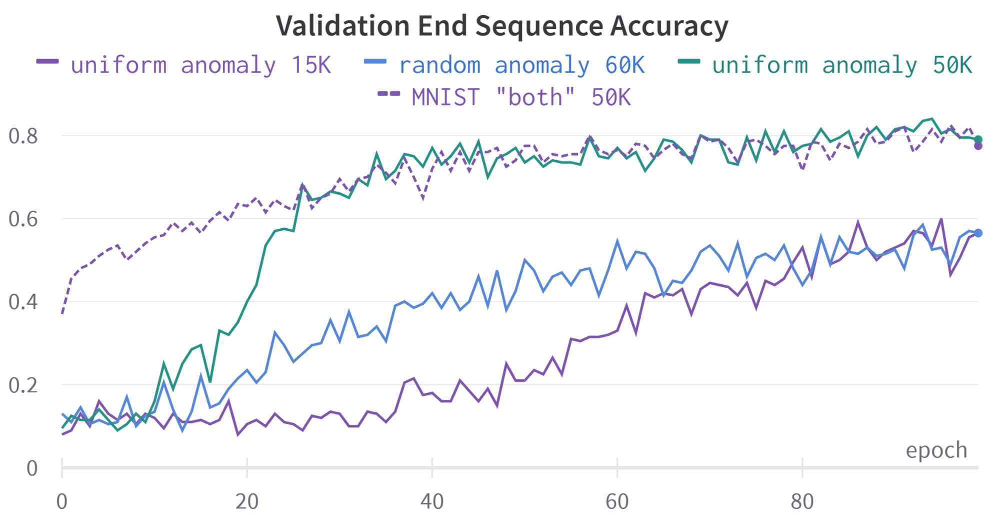


FIGURE 3.19: Relative task difficulty is estimated with sparsity R fixed to 0.2 between the random anomaly with length 9, uniform anomaly with length 9, and MNIST classification with "both" transformation. Using the least parameterized INN that can reliably converge for a given performance level, tasks are compared by number of weights required. The MNIST classification and uniform anomaly detection tasks are able to reach 80% end sequence accuracy at comparable minimal parameterizations of 50,000 weights reflecting equivalent levels of difficulty between the two tasks. The random anomaly detection task converges towards a 57% end sequence accuracy at 60,000 weights. In comparison, similar performance can be achieved on the uniform anomaly detection at 15,000 weights underscoring the significantly more challenging nature of the random anomaly detection task.

As a third possibility, a measure of absolute task difficulty may be established by citing the lowest number of parameters at which performance reliably converges to the ceiling for an INN on that problem. The performance ceiling represents the point at which increasing number of parameters no longer creates a performance increase on non-training data. As shown in Figure 3.20 the absolute task difficulty for the random anomaly data set of sequence length 9 is estimated at 50,000 trainable weights using the methodology described above.

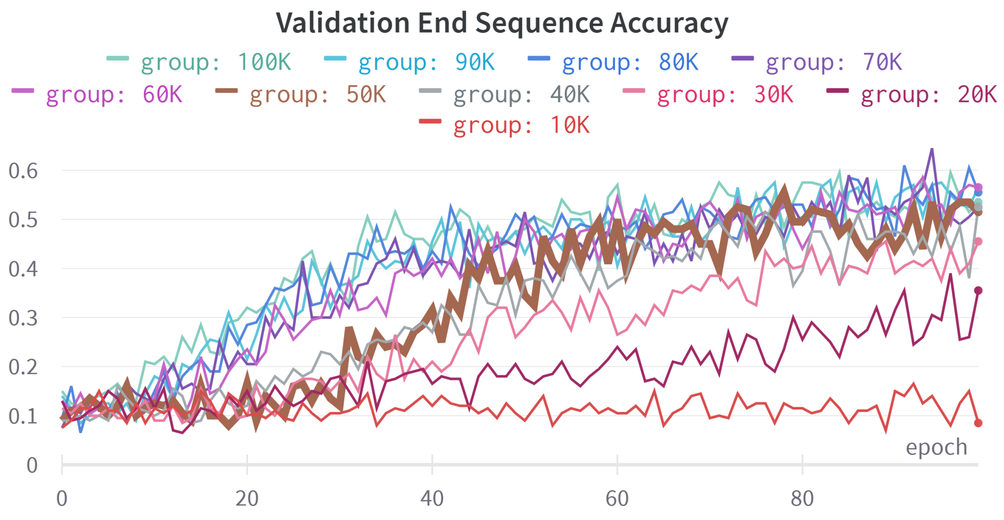


FIGURE 3.20: Absolute task difficulty is estimated with sparsity R fixed to 0.2, on the random anomaly data set with sequence length 9. In this case, validation accuracy converges across various numbers of trainable parameters at an end sequence accuracy level of approximately 55%. The least parameterized INN that can reliably converge with more parameterized networks in these circumstances is plotted in brown representing approximately 50,000 weights which serves as the estimate of task difficulty.

The number of iterations utilized in gauging problem difficulty are left to vary based upon the needs of the data set, being primarily driven by practical considerations. In the case of non-sequential data sets such as those employed with MLP's in basic classification tasks, a minimum of two iterations over the repeated input X must be employed to ensure gradient passes through the full weight space including regions associated with the hidden state H and output Y as shown in Figure 2.5. In a basic example using MNIST classification with the "both" transformation, models are trained at various numbers of iterations as shown in Figure 3.21 using validation end sequence accuracy. As explained, performance lags at one iteration but is comparable for all higher iterations on this specific task. In the case of sequential data sets, sequence length becomes a natural consideration and the number of iterations is left to vary with sequence length and problem characteristics.

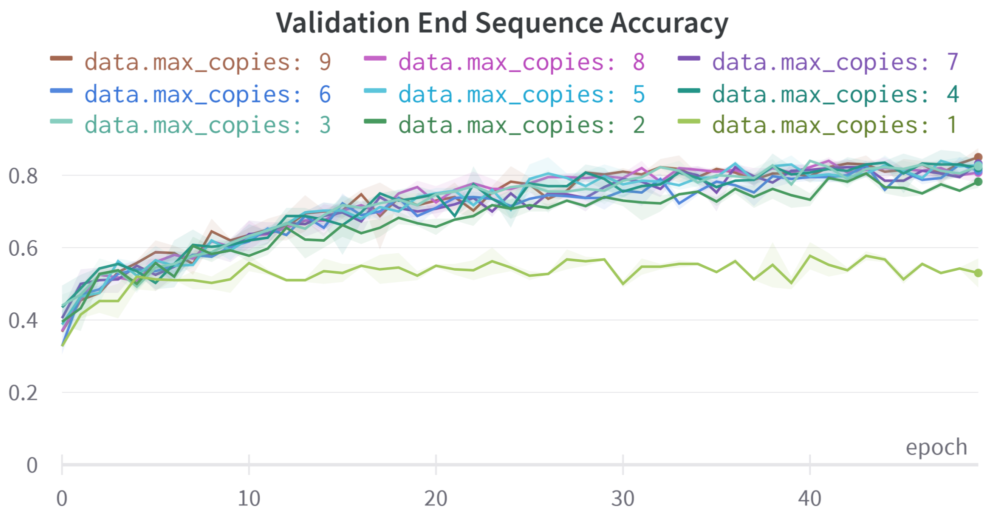


FIGURE 3.21: Training with INN architectures grouped by number of iterations, which was left to vary. At one iteration, performance lagged as gradient does not reach parameters associated with the hidden state. At greater iterations, performance becomes similar between architectures. Training runs were performed on INN models with 100,000 parameters over 50 epochs using MNIST classification with the "both" transformation applied.

An interesting observation arises from the examination of the role of iterations in model behavior. When multi-task learning is employed with the loss function receiving input from each iteration, the models converge towards a fixed point in which the output of a trained model will stabilize even when the number of iterations are increased or decreased from the number employed in training. Importantly, this fixed point behavior was maintained beyond iterations which had been exposed to the loss function. This behavior varies somewhat between traditional layered feed forward model architectures and INN's in that on a typical classification task the INN will converge towards the fixed point at much lower numbers of iterations as gradient passes through more quickly. In the case of the layered feed forward model, the sequential flow of gradient requires the minimum threshold of one iteration per layer must be reached before performance immediately stabilizes at a fixed point.

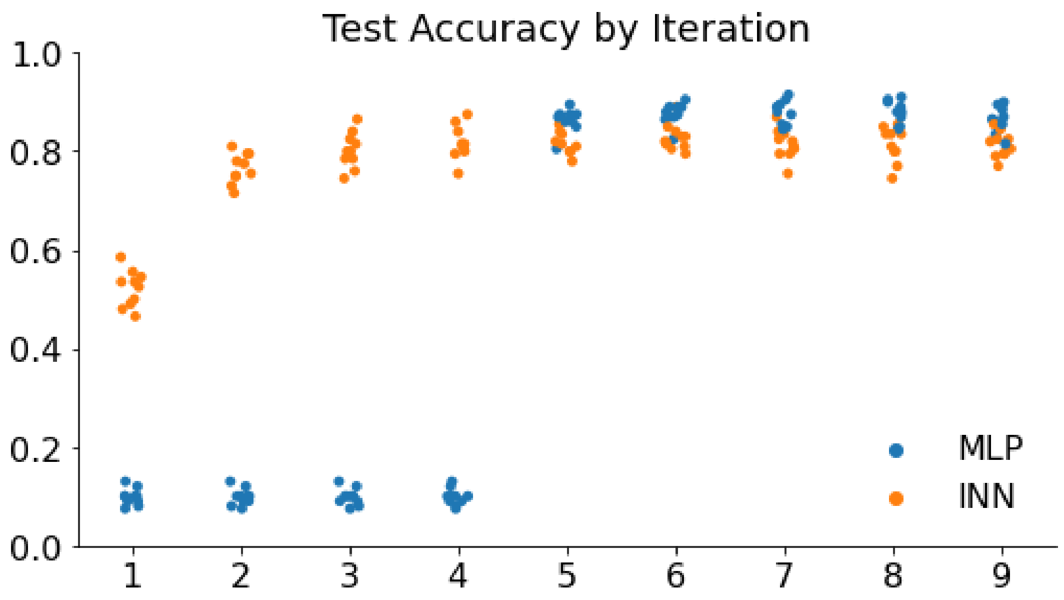


FIGURE 3.22: After training at 5 iterations on the MNIST classification task with the "both" transformation applied using loss captured across each iteration, model performance was gauged on test data at varying iterations. Both MLP and INN architectures appeared to converge to a fixed point in which performance was maintained across varying iterations. When there are fewer iterations than number of layers the sequential flow of gradient in the MLP structure limits this fixed point behavior, as signal from the input has not yet passed through the model and the output is effectively random.

The sparsity and randomized structure of the INN architecture features an inherent flexibility which allows it to scale smoothly with problem difficulty. These characteristics are useful not only in traditional applications which seek to generate solutions, but also provide utility in forming estimates of problem difficulty.

Chapter 4

Conclusions and Discussion

4.1 Summary

Iterative Neural Network (INN) architectures offer a generalized approach to explore neural networks and the role of sparsity in RNN's. Within this framework the research tests various architectural considerations both implied and explicit, exploring the role of architectural restrictions of the weight space on model performance. The results suggest these constraints have little direct impact on performance and may be influencing model behavior through second order impacts on model depth, aspect ratio and sparsity. INN's are demonstrated to carry several performance benefits across a variety of scenarios including improved parameter efficiency and training stability. The primary themes of this body of research are outlined below and summarized in further detail.

1. INN's are a generalized representation of neural networks from which many common variations of neural networks may be derived as special cases.

2. The generalized framework forms a robust approach which performs well across diverse problem sets.

3. Within the INN framework, results demonstrate little direct sensitivity to common architectural considerations such as number and dimension of trainable layers versus randomly assigned weight placements.

4. The interconnected characteristics of model depth, aspect ratio and sparsity appear to demonstrate a more direct link to model performance and stability.

5. The INN architecture may also serve as a mechanism for quantifying problem difficulty owing to its inherent flexibility.

Taken together these findings draw particular focus towards the role of sparse trainable parameter spaces in improving model performance across an array of applications.

4.2 Sparse parameters

Through the use of sparsely arranged trainable model parameters, the results show model performance benefits with lightly parameterized models seeing performance gains as the depth of the parameter space is increased. The effect of greater parameter efficiency may serve to improve performance while limiting the risk of overfitting and benefit applications seeking stronger performance across lighter models. Additionally, model stability improves during training for models as the parameters are arranged with greater sparsity. The effect is essentially a wide performance plateau in which INN's are able to generate comparably high performance across a wider range of hyperparameters including robustness against higher learning rates. Improved stability provides a benefit across a wide range of applications, particularly those which are commonly challenged by instability in training.

Ultimately the conducted research indicates that while typical neural networks feature parameters arranged in dense matrices layers, these arrangements and the hyperparameters associated with them lack a clear advantage in performance. Rather, layers and specified dimensions appear to approximate the combined effects of parameterization and network depth. To this end, the results indicate the use of sparsely arranged weights allow for deeper networks with fewer weights and ultimately more robust training performance with improved stability across a wider array of scenarios. Through the use of sparse blocks of weights to ensure adequate network depth, results demonstrate comparable performance can reliably be achieved across similar networks with a fraction of the total number of trainable weights. The implication here being that in many cases actions as fundamental as increasing total weight counts may simply have been attempts to increase network depth rather than the direct effect of the weights themselves. This act of inefficiently increasing the aggregate number of weights with the indirect goal of increasing network depth may be a primary driver in the often delicate balance between improving model complexity and overfitting the training data. It would appear that the specification of number of layers and their dimensions attempts to strike a balance between the expansion of model depth and the associated increase in trainable parameters.

4.3 Further research

INN's provide a rich framework for the study of traditional neural network architectures with many vectors for future research. One direction of further research may be examining the overriding principles behind optimal weight space sparsity and parameterization, potentially aligning with the concepts of percolation theory. Another area of special interest may be the application of INN's in benchmarking quantifiable problem difficulty as a metric across varying tasks as demonstrated in Section 3.4.4. Additionally, the adaptability of INN's in their general form across multiple problem types appears well suited to meta-learning applications.

4.4 Final conclusions

Overall, this thesis develops the initial INN framework and associated implications for traditional model architectures while creating space for further research. The framework of understanding RNNs as a broader class of neural networks within which feed forward networks exist as a special case does not appear to be widely appreciated. Additionally, the diminished impact of typical architecture design in network specification in favor of randomized weight distribution is both surprising and runs counter to conventional wisdom across the deep learning field. Lastly, the narrowed focus on the interrelated metrics of network depth, aspect ratio and sparsity provide an alternative for machine learning practitioners as they contemplate network design.

Bibliography

[1] Coryn A L Bailer-Jones, David J C MacKay, and Philip J Withers. "A recurrent neural network for modelling dynamical systems". In: *Network: Computation in Neural Systems* 9.4 (Nov. 1998), p. 531. DOI: 10.1088/0954-898X/9/4/008. URL: https://dx.doi.org/10.1088/0954-898X/9/4/008.

[2] G. Bebis and M. Georgiopoulos. "Feed-forward neural networks". In: *IEEE Potentials* 13.4 (1994), pp. 27–31. DOI: 10.1109/45.329294.

[3] A. Bhaya and E. Kaszkurewicz. "Iterative methods as dynamical systems with feedback control". In: *42nd IEEE International Conference on Decision and Control (IEEE Cat. No.03CH37475)*. Vol. 3. 2003, 2374–2380 Vol.3. DOI: 10.1109/CDC.2003.1272974.

[4] Lukas Biewald. *Experiment Tracking with Weights and Biases*. Software available from wandb.com. 2020. URL: https://www.wandb.com/.

[5] Thomas M. Breuel. *Benchmarking of LSTM Networks*. 2015. DOI: 10.48550/ARXIV.1508.02774. URL: https://arxiv.org/abs/1508.02774.

[6] Hyunghun Cho et al. "Basic Enhancement Strategies When Using Bayesian Optimization for Hyperparameter Tuning of Deep Neural Networks". In: *IEEE Access* 8 (2020), pp. 52588–52608. DOI: 10.1109/ACCESS.2020.2981072.

[7] Michael Crawshaw. *Multi-Task Learning with Deep Neural Networks: A Survey*. 2020. DOI: 10.48550/ARXIV.2009.09796. URL: https://arxiv.org/abs/2009.09796.

[8] G. Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals, and Systems (MCSS)* 2.4 (Dec. 1989), pp. 303–314. ISSN: 0932-4194. DOI: 10.1007/BF02551274. URL: http://dx.doi.org/10.1007/BF02551274.

[9] Jesse Dodge et al. *RNN Architecture Learning with Sparse Regularization*. 2019. DOI: 10.48550/ARXIV.1909.03011. URL: https://arxiv.org/abs/1909.03011.

[10] William Falcon and The PyTorch Lightning team. *PyTorch Lightning*. Version 1.4. Mar. 2019. DOI: 10.5281/zenodo.3828935. URL: https://github.com/Lightning-AI/lightning.

[11] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[12] Yann LeCun, Corinna Cortes, Chris Burges, et al. *MNIST handwritten digit database*. 2010.

[13] Lizhi Liao et al. "An Empirical Study of the Impact of Hyperparameter Tuning and Model Optimization on the Performance Properties of Deep Neural Networks". In: *ACM Trans. Softw. Eng. Methodol.* 31.3 (Apr. 2022). ISSN: 1049-331X. DOI: 10.1145/3506695. URL: https://doi.org/10.1145/3506695.

[14] Shiwei Liu et al. *Efficient and effective training of sparse recurrent neural networks*. 2021. DOI: 10.1007/s00521-021-05727-y. URL: https://doi.org/10.1007/s00521-021-05727-y.

[15] TorchVision maintainers and contributors. *TorchVision: PyTorch's Computer Vision library*. Nov. 2016. URL: https://github.com/pytorch/vision.

[16] B. B. Mandelbrot. *The fractal geometry of nature*. 3rd ed. New York: W. H. Freeman and Comp., 1983.

[17] Benoit Mandelbrot. "How Long Is the Coast of Britain? Statistical Self-Similarity and Fractional Dimension". In: *Science* 156.3775 (1967), pp. 636–638. ISSN: 00368075, 10959203. URL: http://www.jstor.org/stable/1721427 (visited on 03/17/2023).

[18] Benoit B. Mandelbrot. "FRACTAL ASPECTS OF THE ITERATION OF z to z(1- z) FOR COMPLEX AND z". In: *Annals of the New York Academy of Sciences* 357.1 (1980), pp. 249–259. DOI: https://doi.org/10.1111/j.1749-6632.1980.tb29690.x. eprint: https://nyaspubs.onlinelibrary.wiley.com/doi/pdf/10.1111/j.1749-6632.1980.tb29690.x. URL: https://nyaspubs.onlinelibrary.wiley.com/doi/abs/10.1111/j.1749-6632.1980.tb29690.x.

[19] Mina Moradi Kordmahalleh, Mohammad Gorji Sefidmazgi, and Abdollah Homaifar. "A Sparse Recurrent Neural Network for Trajectory Prediction of Atlantic Hurricanes". In: *Proceedings of the Genetic and Evolutionary Computation Conference 2016*. GECCO '16. Denver, Colorado, USA: Association for Computing Machinery, 2016, pp. 957–964. ISBN: 9781450342063. DOI: 10.1145/2908812.2908834. URL: https://doi.org/10.1145/2908812.2908834.

[20] Fionn Murtagh. "Multilayer perceptrons for classification and regression". In: *Neurocomputing* 2.5 (1991), pp. 183–197. ISSN: 0925-2312. DOI: https://doi.org/10.1016/0925-2312(91)90023-5. URL: https://www.sciencedirect.com/science/article/pii/0925231291900235.

[21] Sharan Narang, Eric Undersander, and Gregory F. Diamos. "Block-Sparse Recurrent Neural Networks". In: *CoRR* abs/1711.02782 (2017). arXiv: 1711.02782. URL: http://arxiv.org/abs/1711.02782.

[22] Patrick Neary. "Automatic Hyperparameter Tuning in Deep Convolutional Neural Networks Using Asynchronous Reinforcement Learning". In: *2018 IEEE International Conference on Cognitive Computing (ICCC)*. 2018, pp. 73–77. DOI: 10.1109/ICCC.2018.00017.

[23] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training recurrent neural networks". In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, June 2013, pp. 1310–1318. URL: https://proceedings.mlr.press/v28/pascanu13.html.

[24] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[25] George Philipp, Dawn Song, and Jaime G. Carbonell. "Gradients explode - Deep Networks are shallow - ResNet explained". In: *CoRR* abs/1712.05577 (2017). arXiv: 1712.05577. URL: http://arxiv.org/abs/1712.05577.

[26] Andrinandrasana David Rasamoelina, Fouzia Adjailia, and Peter Sinčák. “A Review of Activation Function for Artificial Neural Network”. In: *2020 IEEE 18th World Symposium on Applied Machine Intelligence and Informatics (SAMI)*. 2020, pp. 281–286. DOI: 10.1109/SAMI48414.2020.9108717.

[27] Alexander Rehmer and Andreas Kroll. “On the vanishing and exploding gradient problem in Gated Recurrent Units”. In: *IFAC-PapersOnLine* 53.2 (2020). 21st IFAC World Congress, pp. 1243–1248. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2020.12.1342. URL: https://www.sciencedirect.com/science/article/pii/S2405896320317481.

[28] Battelle Rencontres and Jurgen Moser. *Dynamical systems, theory and applications / edited by J. Moser*. English. Springer-Verlag Berlin, 1975, vi, 624 p. ISBN: 3540071717.

[29] Sebastian Ruder. “An Overview of Multi-Task Learning in Deep Neural Networks”. In: *CoRR* abs/1706.05098 (2017). arXiv: 1706.05098. URL: http://arxiv.org/abs/1706.05098.

[30] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning representations by back-propagating errors”. In: *nature* 323.6088 (1986), pp. 533–536.

[31] Haşim Sak, Andrew Senior, and Françoise Beaufays. *Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition*. 2014. arXiv: 1402.1128 [cs.NE].

[32] Hojjat Salehinejad et al. *Recent Advances in Recurrent Neural Networks*. 2018. DOI: 10.48550/ARXIV.1801.01078. URL: https://arxiv.org/abs/1801.01078.

[33] Alex Sherstinsky. “Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network”. In: *Physica D: Nonlinear Phenomena* 404 (Mar. 2020), p. 132306. DOI: 10.1016/j.physd.2019.132306.

[34] Toshiharu Sugie and Toshiro Ono. “An iterative learning control law for dynamical systems”. In: *Automatica* 27.4 (1991), pp. 729–732. ISSN: 0005-1098. DOI: https://doi.org/10.1016/0005-1098(91)90066-B. URL: https://www.sciencedirect.com/science/article/pii/000510989190066B.

[35] Rhian Taylor et al. “Sensitivity Analysis for Deep Learning: Ranking Hyperparameter Influence”. In: *2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)*. 2021, pp. 512–516. DOI: 10.1109/ICTAI52525.2021.00083.

[36] Christian V. *Chaos Theory & Double Pendulum - 3.jpg*. [Online; accessed 2-March-2023]. 2014. URL: https://upload.wikimedia.org/wikipedia/commons/1/17/Chaos_Theory_%5C%26_Double_Pendulum_-_3.jpg.

[37] Yong Yu et al. “A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures”. In: *Neural Computation* 31.7 (July 2019), pp. 1235–1270. ISSN: 0899-7667. DOI: 10.1162/neco_a_01199. eprint: https://direct.mit.edu/neco/article-pdf/31/7/1235/1053200/neco_a_01199.pdf. URL: https://doi.org/10.1162/neco%5C_a%5C_01199.