DYNAMIC ANALYSIS OF RECURRENT NEURAL NETWORKS

A Dissertation

by

HAN WANG

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2020

Major Subject: Computer Science

ABSTRACT

With the advancement in deep learning research, neural networks have become one of the most powerful tools for artificial intelligence tasks. More specifically, recurrent neural networks (RNNs) have achieved state-of-the-art in tasks such as hand-writing recognition and speech recognition. Despite the success of recurrent neural networks, how and why do neural nets work is still not sufficiently investigated. My work on the dynamical analysis of recurrent neural networks can help understand how the input features are extracted in the recurrent layer, how the RNNs make decisions, and how the chaotic dynamics of RNNs affects its behaviors. Firstly, I investigated the dynamics of recurrent neural networks as autonomous dynamical system in the experiment of a two-joint limb controlling task and compared the empirical result and the theoretical analysis. Secondly, I investigated the dynamics of non-autonomous recurrent neural networks on two benchmark tasks: sequential MNIST recognition task and DNA splice junction classification task. How the hidden states of long-short term memory (LSTM) and gated recurrent unit (GRU) cells learn new features and how the input sequence is extracted are demonstrated with experiments. Finally, based on the understanding of the external and internal dynamics of recurrent units, I proposed several algorithms for recurrent neural network compression. The algorithms demonstrate reasonable performance in compression ratio and are able to sustain the performance of the original models.

DEDICATION

To my family.

To Ping.

ACKNOWLEDGMENTS

Thanks to Qinbo Li for monthly research discussion. Thanks to my team leader Dr. Boon Loong Ng, Dr. Wenxun Qiu and all my colleagues at Samsung Research America for supporting my study.

CONTRIBUTORS AND FUNDING SOURCES

This work was supported by a dissertation committee consisting of Professor Yoonsuck Choe, Professor Theodora Chaspari and Professor Tracy Hammond of the Department of Computer Science and Engineering and Professor Mi Lu of the Department of Electrical Engineering.

All work for the dissertation was completed independently by the student.

There are no outside funding contributions to acknowledge related to the research and compilation of this document.

NOMENCLATURE

TAMU	Texas A&M University
ANN	Artificial Neural Network
AI	artificial intelligence
AGI	artificial general intelligence
RNN	Recurrent Nerual Network
NEAT	Neuro-evolution with Augmenting Topology
BPTT	Back-propagation through time
LSTM	Long-shot Term Memory
GRU	Gated Recurrent Unit
ReLU	Rectified Linear Unit
IRNN	Identity Recurrent Neural Network
URNN	Unitary Recurrent Neural Network
DoF	degree of freedom
WC	world-centered sensory representation
AC	agent-centered sensory representation
RAC	relative agent-centerd sensory representation
HMM	Hidden Markov Model
DNA	deoxyribonucleic acid
KL-divegence	Kullback-Leibler divergence
PCA	principal component analysis
t-SNE	t-distributed stochastic neighbor embedding
PSNR	Peak Signal-to-Noise Ratio

CNN	convolutional nerual network

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

1. INTRODUCTION AND LITERATURE REVIEW

This dissertation consists of five chapters. Literature review is the first chapter that summarizes the existing knowledge of artificial neural networks and the related research challenges. The second chapter is dedicated to my work on the dynamical analysis of autonomous recurrent neural networks, and the third chapter is dedicated to my work on the analysis of non-autonomous recurrent neural networks. Chapter 4 demonstrates an application of the dynamical analysis study — compression of recurrent neural networks. Chapter 5 concludes with the contribution of my work.

1.1 A Brief History of Artificial Neural Networks

The early study of artificial neural networks could date back to 1940s. Inspired by the biological neural networks, McCulloch et al. [5] implemented a network of threshold logic units as a novel computational model. In 1949, the Hebbian learning throey was proposed by Hebb [6], based on the observation neuroplasticity. Hebbian learning theory later became the neuronal basis of unsupervised learning. The Hopfield network [7] and Boltzmann machine [8] were developed based on the Hebbian learning theory. In 1974, Werbos introduced back-propagation to the training of neural networks [9]. Back-propagation has become a popular training algorithm of neural networks. Due to its rigorous mathematical characteristics, the back-propagation algorithm is efficient in optimizing the loss function of small neural networks. In 2006, Hinton et al. established the "deep learning" era of artificial neural networks by demonstrating how a deep belief network [10][11] can be trained. Since then, deep neural networks have shown great capability on various applications in artificial intelligence.

1.2 Recent Trends on Artificial Neural Networks Research

There are numerous open questions in the field of artificial neural network research. On the theory side, new architectures and new optimization algorithms still will be the main direction of research [12][13]. Besides, the model's interpretability [14] has also been considered important. On the application side, machine comprehension is a challenging topic for computer vision [15] and natural language processing [16]. Multi-task learning also draws broad attention from researchers [17][18]. In terms of the machine learning engineering aspect, the emerging demand for more efficient development and computation has led to research on model compression [19][20] and neural architecture search [21][22].

1.3 Recurrent Neural Networks

Artificial neural networks can be classified into feed-forward neural networks and recurrent neural networks (RNN) based on the network topology. Feed-forward neural networks only allow signal to travel one way, and directly associate the present input to the output. The RNNs allow signal to propagate in loop(s). Because of the loop(s), the RNNs generate output as fusion of the present input and hidden states that encode the past input.

Similar to feed-forward neural networks, a recurrent neural network (RNN) usually has three component layers: an input layer, hidden layer(s) and an output layer. The difference lies in the existence of recurrent connections in the hidden layer(s).

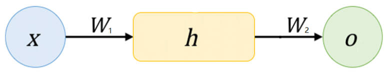


Figure 1.1: Schematic of feed-forward neural network

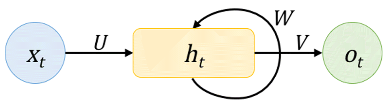


Figure 1.2: Schematic of recurrent neural network

As shown in Figure 1.2, the output o_t is calculated based on the following equations:

$$\vec{o_t} = \sigma_o(V\vec{h_t}) \tag{1.1}$$

$$\vec{z_t} = W\vec{h_{t-1}} + U\vec{x_t} + b \tag{1.2}$$

$$\vec{h_t} = \sigma_h(\vec{z_t}) \tag{1.3}$$

where U, V, W are connection weight matrices and σ_o, σ_h are non-linear activation functions. Since $\vec{h_t}$ depends on the prior hidden state $\vec{h_{t-1}}$, the RNN exhibits internal memory to time sequence input.

1.3.1 Training of Recurrent Neural Networks

The training process of recurrent neural network is similar with the training process of feed-forward neural network. There are gradient-based algorithms and gradient-free algorithms.

1.3.1.1 Back-propagation Through Time

One commonly used gradient-based algorithm is back-propagation through time (BPTT) [23], which applies back-propagation on an unrolled recurrent neural network. The dynamics of hidden layer is determined by how the hidden neurons are wired as well as how the weights are tuned. However, training RNN with gradient can be difficult since the long propagation of signals in hidden layer(s) usually results in gradient exploding/vanishing problem.

To illustrate the gradient exploding/vanishing problem, suppose the loss function is L:

$$\frac{\partial L}{\partial h_1} = \frac{\partial L}{\partial h_T}\frac{\partial h_T}{\partial h_1} = \frac{\partial L}{\partial h_T}\prod_{k=1}^{T-1}\frac{\partial h_{k+1}}{\partial h_k} = \frac{\partial L}{\partial h_T}\prod_{k=1}^{T-1}(\sigma_h'(z_k)W) \tag{1.4}$$

When $||\sigma_h'(z_k)W|| \neq 1$ and T is large enough, the norm of the product can either become very large or close to zero.

There are several ways to resolve the gradient exploding/vanishing problem. A popular approach is gradient clipping, which is widely used in encapsulated recurrent cells, such as long-short term memory(LSTM) [24] and gated recurrent unit (GRU) [25]. Some other approaches attempt to stabilize the gradient with $||\sigma'_h(z_k)W|| \approx 1$, including ReLU with identity RNN (IRNN) [26] and unitary RNN (URNN) [27].

Besides these gradient-based methods, Gradient-free optimization such as evolutionary algorithm (e.g.neuro-evolution with augmenting topology [28]) has been proven to be effective in training RNNs.

1.3.1.2 Neuro-evolution

Neuro-evolution [29] is a representative gradient-free method for training recurrent neural networks. It applies evolutionary algorithm to the RNN parameters by encoding the parameters into genotypes and defining a fitness function. An improved version of neuro-evolution NEAT[28] can evolve the topology of the neural network while training, which has been proved to be effective in solving reinforcement learning tasks [30].

Compared with gradient-based methods, neuro-evolution is less likely to get stuck to local optima, but requires stronger computational power to handle the large search space for the evolutionary algorithm. Recent advancements in hardware have made deep neuro-evolution [31][32] possible to achieve competitive results compared to gradient-based methods for large-scale neural networks.

1.3.2 Vanilla Recurrent Neural Networks

Vanilla recurrent neural networks refer to the RNNs that do not have regular topological substructures. These recurrent neural networks often exhibit complicated dynamics and are difficult to train. The identity RNN (IRNN) [26] and unitary RNN (URNN) [27] initialize the weights to be orthogonal or unitary to restrict the internal dynamics in a small sub-state space and add extra computational cost to maintain such restrictions.

Alternatively, some vanilla models (e.g.liquid state machines [33] and echo state networks [34]) utilize the idea of reservoir computing [35] and contain randomized fixed recurrent structures and only the linear output layer is trained. Although the reservoir computing algorithm has appeared to be successful in several machine learning tasks [36] and shown its advantage in hardware implementation [37], it encapsulates the recurrent layer as a complete black box and creates difficulties in understanding how and why the model works.

1.3.3 Long-shot Term Memory (LSTM) and Gated Recurrent Units (GRU)

The Vanilla RNNs do not perform well on remembering long sequences. Long-short term memory (LSTM) [24] was designed to memorize information over long time periods with its insensitivity to temporal gaps between information. The central idea behind the LSTM architecture is a memory cell which can maintain its state over time, and non-linear gating units which regulate the information flow into and out of the cell [38].

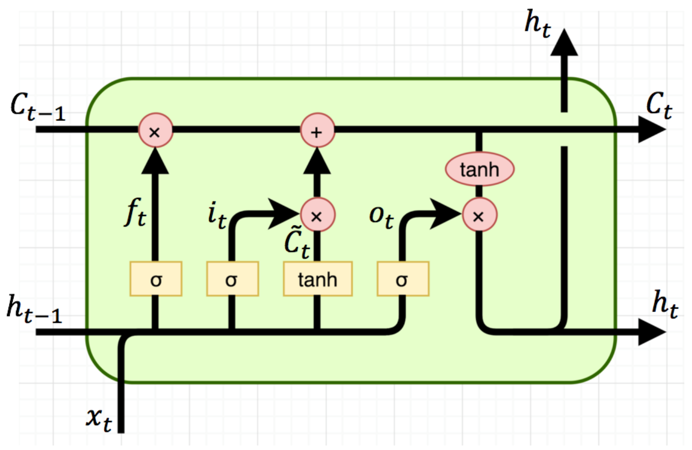


Figure 1.3: Schematic of LSTM cell. Reprinted from *Understanding LSTM Networks* by Christopher Olah.[1]

The update rules of LSTM cells can be represented as:

$$\begin{aligned}
\text{forget gate:}\quad & \vec{f_t} = \sigma_g(W_f\vec{x_t} + U_f\vec{h_{t-1}} + \vec{b_f}) \\
\text{input gate:}\quad & \vec{i_t} = \sigma_g(W_i\vec{x_t} + U_i\vec{h_{t-1}} + \vec{b_i}) \\
\text{output gate:}\quad & \vec{o_t} = \sigma_g(W_o\vec{x_t} + U_o\vec{h_{t-1}} + \vec{b_o}) \\
\text{cell state:}\quad & \vec{c_t} = \vec{f_t} \circ \vec{c_{t-1}} + \vec{i_t} \circ \sigma_c(W_c\vec{x_t} + U_c\vec{h_{t-1}} + \vec{b_c}) \\
\text{hidden state:}\quad & \vec{h_t} = \vec{o_t} \circ \sigma_h(\vec{c_t})
\end{aligned} \tag{1.5}$$

Another popular variant of gated units is the gated recurrent unit (GRU) [25]. Compared with LSTM cell, the GRU cell does not contain an output gate, so that the hidden states are exposed without hidden memory unit. The GRU has less parameters and is computational efficient.

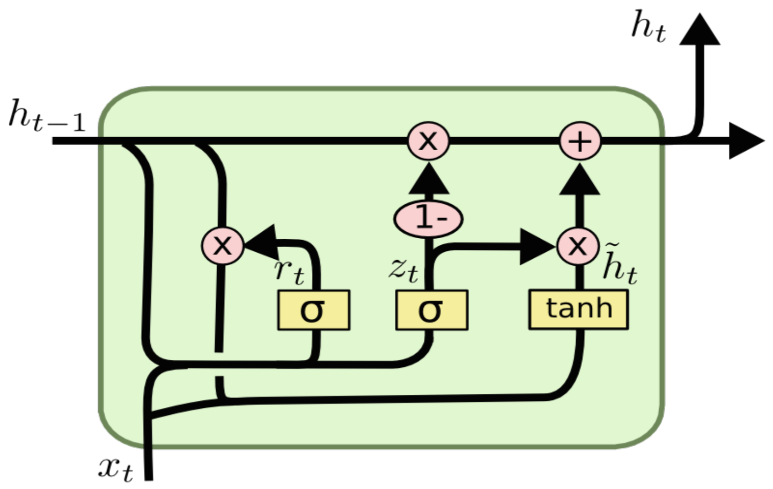


Figure 1.4: Schematic of GRU cell. Reprinted from *What is a Recurrent Neural Networks (RNNS) and Gated Recurrent Unit (GRUS)*. [2]

The update rules of GRU cells can be represented as:

$$\begin{aligned}
\text{update gate:} \quad & \vec{z_t} = \sigma_g(W_z\vec{x_t} + U_z\vec{h_{t-1}} + \vec{b_z}) \\
\text{reset gate:} \quad & \vec{r_t} = \sigma_g(W_r\vec{x_t} + U_r\vec{h_{t-1}} + \vec{b_r}) \\
\text{hidden state:} \quad & \vec{h_t} = (1 - \vec{z_t}) \circ \vec{h_{t-1}} + \vec{z_t} \circ \sigma_h(W_h\vec{x_t} + U_h(\vec{r_t} \circ \vec{h_{t-1}}) + \vec{b_h})
\end{aligned} \tag{1.6}$$

LSTM and GRU are both popular used in modern deep learning since the model is highly scalable and convenient for GPU acceleration.

1.4 Recurrent Neural Networks as Dynamical Systems

A recurrent neural network computes iteratively:

$$h_t = f(f(f(...f(h_0, x_1)..., x_{t-2}), x_{t-1}), x_t) \tag{1.7}$$

Simple functions can become complicated when applied with iterations. For example, Mandelbrot set M, $z_{n+1} = z_n^2 + c$ for $c \in M$ and the logistic map $x_{n+1} = rx_n(1 - x_n)$.

A dynamical system describes the rule of time evolution of vector(s) in a vector space. RNNs are discrete-time dynamical systems. The state space of RNNs can be represented as the activation status of neurons.

A time-discrete dynamical system can be either autonomous or non-autonomous. The update rule of an autonomous dynamical system is

$$x_{t+1} = f(x_t) \qquad \forall t \in N \tag{1.8}$$

the next state only depends on the current state. Therefore, the sequence of $\{x_t\}$ only depends on x_0. The update rule of a non-autonomous dynamical system is

$$x_{t+1} = f(x_t, t) \qquad \forall t \in N \tag{1.9}$$

the next state depends on both the current state and tetime.

For RNNs, if the input is fully observable by the network, for instance, controlling a robot arm in a static environment, or there's no input/zero input, then the RNNs turn out to have autonomous dynamic property. If the input is partially observable, for instance, to forecast stock prices, the RNNs are non-autonomous.

1.5 Summary

Starting from the imitation of biological neural networks, artificial neural networks were designed to mimic the learning and reasoning of intelligent beings through computation. Nowadays, with efficient training algorithms and development frameworks [39][40][41], highly scalable connection topology and the fast rising computational power of hardwares, the cost of deploying large-scale artificial neural network models is becoming lower and lower.

As one kind of the artificial neural networks, the recurrent neural networks are able to learn sequential data with its internal memory. Due to the recurrence in the connection topology, the recurrent neural networks exhibit complicated dynamics. The black box property of recurrent neural networks has made training and model analysis to be challenging. Despite more powerful gated units such as long-short term memory (LSTM) and gated recurrent unit (GRU) retrain such difficulty, our understanding of the feature extraction process and the learning mechanism in recurrent neural networks is still very limited. Decoding the black box can possibly help improve the machine learning model design and get us a step closer to artificial general intelligence (AGI).

Since the recurrent neural networks are dynamical systems, analyzing recurrent neural networks as dynamical systems provides some mathematical insights for the quantification of stability and instability, which can help understand the dynamics and the learning procedure of recurrent neural networks.

2. ANALYSIS OF AUTONOMOUS RECURRENT NEURAL NETWORKS [*]

Unlike feedforward neural networks, a recurrent neural network (RNN) allows neurons to send feedback signals which create an internal memory state of the network. As the topological complexity grows, the diversity of the temporal behaviors of the network increases dramatically. This diversity makes RNN applicable to time series problems in many different categories such as robot control [42] and speech recognition [43].

A recurrent neural network can usually be modeled as a non-linear dynamical system. Beer pioneered dynamical approaches to cognitive science. In 1995, he introduced the use of the language of dynamical system theory as a general theoretical framework for the synthesis and analysis of autonomous agents controlled by neural networks [44][45]. However, even though non-linear dynamical systems have been studied for more than 100 years, there are still many parts that are not well understood. For example, Hilbert's 16th problem which is about the determination of the upper bound for the number of limit cycles and their relative locations, remains unsolved [46].

The simplest type of dynamical system is autonomous dynamical system. The dynamics of autonomous dynamical systems take place on smooth manifolds. Although it is difficult to mathematically formalize the RNN models into ordinary differential equations as some classic dynamical systems, we can still attempt to analyze and visualize the local dynamics, in order to reveal how do RNNs work, when do they succeed and when do they fail.

Controllers for tool-use may require RNNs. The use of tools has been extensively studied and is still an attractive topic for scientists from both neuroscience and robotics. The use of tools in animals indicates high levels of cognitive capability, and, aside from humans, is observed only in a small number of higher mammals and avian species [47, 48, 49, 50]. Chung and Choe [51] showed that simple neural circuits can in fact be evolved to use the simplest form of tool, i.e., a "bread-crumb" dropped in the environment to serve as external memory. But it is yet unknown

[*]© 2016 IEEE. Reprinted, with permission, from Han Wang, et al. Dynamical analysis of recurrent neural circuits in articulated limb controllers for tool use.

how such a capability could have emerged from simple organisms. Li et al. [52] evolved RNN controllers using NEAT for target reaching tasks with a tool. The NeuroEvolution of Augmenting Topologies (NEAT) algorithm by Stanley and Miikkulainen [28] is an evolutionary algorithm for evolving RNNs of arbitrary topology. My empirical study of autonomous RNNs are based on the RNN controller used in the two-joint arm controlling task. Part of this Chapter was previously published as [3] and is used here with permission from IEEE.

2.1 Approach

2.1.1 Overview

The neural circuit works as a controller for a two degree-of-freedom articulated limb and the task is to reach a target which is out of the range of the arm alone but reachable if the arm grabs a tool (Fig. 2.1 and 2.2). The NEAT algorithm creates generalized RNNs which have arbitrary topology. The RNNs can evolve in all aspects: number of neurons, connectivity and weights of connection. The activation function in the hidden and output neurons is a sigmoid function, making the network exhibit rich non-linear dynamics. Since the scale and complexity of the RNNs increases as they evolve, we limited the number of generations to keep the complexity of the RNNs under control.

Firstly, we evaluated the performance of the RNN controller. The correlation of success rate and many parameters such as the starting angles of the arm, and different locations of the tool and the target. Given all the initial values of the input vector, the results are deterministic, however, the results show irregular distributions dependent on some parameters.

Next we examined the neural activity in different parametric trials. We find that the behavior of the arm is encoded in the activities of the neurons. As a non-autonomous dynamical system, in successful trials (target reached), the set of hidden neurons activities approach a stable state when the arm reaches the target. In the failed trials, the hidden neurons fall into many kinds of limit cycles so the arm either oscillates or gets stuck before reaching the target.

2.1.2 Evolving the Neural Circuits Using NEAT

2.1.2.1 Setup of the Task

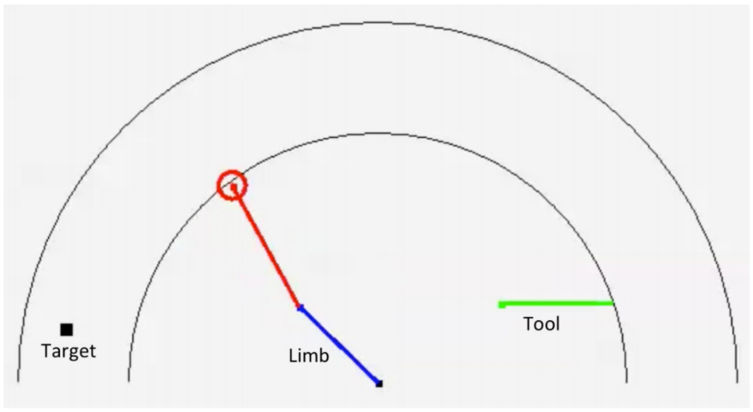
The task is to control a two-limbed articulated arm in an object reaching task. The degree of freedom of the arm is 2, and the arm moves on a 2-D plane. The environment was equipped with a tool (a stick) that can be picked up and used to reach objects beyond the range of the limb (Fig. 2.1).

2.1.2.2 Input and Output

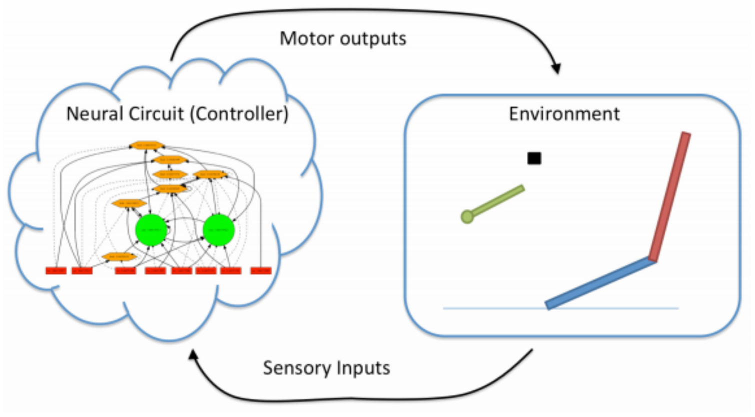
It is important to define the proper sensory inputs to represent the environment properly, especially for artificial agents learning through action and feedback loops [47]. In [53], the authors compared two forms of representations: world-centered sensory representation (WC) and agent-centered sensory representation (AC). AC can use polar coordinates while WC can use Cartesian coordinates. The author further proposed the relative agent-centered sensory representation (RAC) which is the relative difference between two ACs. In our experiment, we use RAC as the sensory inputs to the limb controller neural circuit. The details of AC and RAC are as follows:

$$\begin{aligned}
&AC_{\mathit{end\_}}eff = (\phi_1, d_1)\\
&AC_{target} = (\phi_2, d_2)\\
&AC_{tool} = (\phi_3, d_3)\\
&RAC_{\mathit{end\_}}eff\&target = (\phi_2 - \phi_1, d_2 - d_1)\\
&RAC_{\mathit{end\_}}eff\&tool = (\phi_3 - \phi_1, d_3 - d_1)
\end{aligned} \tag{2.1}$$

The input layer includes both perceptual and proprioceptive neurons [54]. 6 input neurons are needed: two joint angles θ_1 and θ_2, relative distance and angle to the target $\phi_2 - \phi_1$ and $d_2 - d_1$, relative distance and angle to the tool $\phi_3 - \phi_1$, $d_3 - d_1$. (Fig. 2.2a) The output layer serves as motor

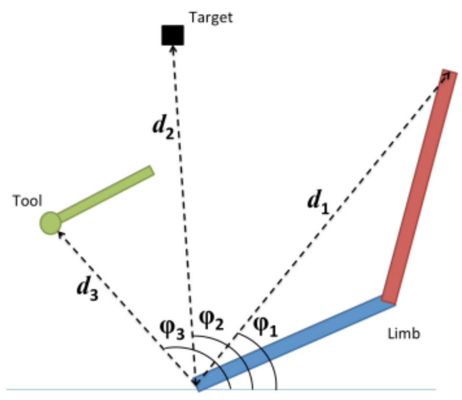


(a) Example Task Condition

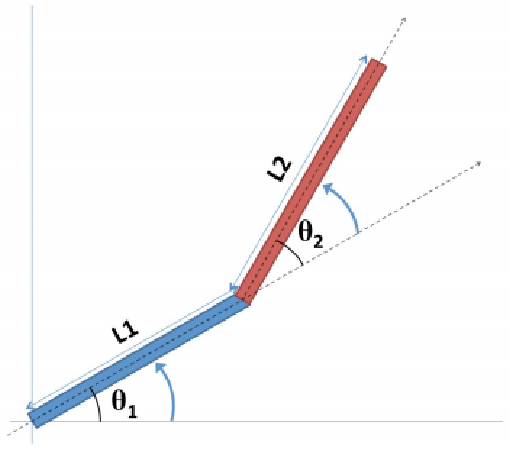


(b) Controller-environment Interaction

Figure 2.1: The task and the environment. (a) The environment consists of a two degree-of-freedom articulated limb, a target object, and a tool, all on a 2D plane. Note that the tool's initial locations are variable. The two half-circles indicate the original reach (inner) and the reach with tool use (outer). (b) The interaction cycle between the neural circuit controller and the environment. When the end-effector reaches the tool handle, the tool is automatically attached to the limb and the tip of the tool becomes the end effector. Note: the controller is not explicitly notified of the limb extension due to the tool pickup. Reprinted with permission from [3].



(a) Sensory Representation



(b) Kinematics

Figure 2.2: Agent-centered sensory representation (AC) and kinematics of the joint arm. (a) AC uses a polar coordinate system. ϕ_1, ϕ_2 and ϕ_3 represent the angles for the end effector, the target and the tool, respectively. d_1, d_2 and d_3 indicate the distances to the end effector, the target, and the tool, respectively. (b) The limb consists of two joints θ_1 and θ_2, and the arm segments $L1$ and $L2$ (with the length ratio of $L1 : L2 = 1 : 1.25$). The two joint angles θ_1 and θ_2 are controlled by the neural circuit output. Reprinted with permission from [3].

neurons to control the change of two joint angles $\Delta\theta_1$ and $\Delta\theta_2$ with maximum step size of $1.5°$ (Fig. 2.2b).

We also use joint limit detectors as two additional input neurons:

$$v_{\{\theta_1,\theta_2\}} = \begin{cases} 1 & \text{if}\{\theta_1, \theta_2\} \geq 150° \\ -1 & \text{if}\{\theta_1, \theta_2\} < 150° \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

Overall, 8 input neurons and 2 output neurons are used in the neural circuit, and these are the only neurons in the initial network (i.e., no hidden neurons).

2.1.2.3 Fitness Criteria

We tested three basic fitness criteria and their different combinations to evolve the neural circuit controller. For each controller, by the end of 100 trials, the following quantities were calculated:
(1) D: distance between the end effector and the target;
(2) S: steps taken to reach the target; and
(3) T: tool pick-up frequency.

$$D = 1 - \frac{\sum_k \|\vec{o_k} - \vec{e_k}\|}{K \times D_{max}} \tag{2.3}$$

$$S = 1 - \frac{\sum_k S_k}{K \times S_{max}} \tag{2.4}$$

$$T = 1 - \frac{\sum_k t_k}{K}, t_k = \begin{cases} 1 & \text{if tool is picked up} \\ 0 & \text{otherwise} \end{cases} \tag{2.5}$$

where $\vec{o_k}$ and $\vec{e_k}$ are the coordinates of the target object and the end effector of the limb, respectively. The Euclidean distance ($L2$ norm) $\| \circ \|$ is normalized by the maximum radius of the environment D_{max}. K indicates the total number of the trials ($K = 100$ in the experiment) and k indicates kth trial. s_k indicates the number of steps taken before reaching the target, and S_{max} is the maximum movement steps for each trial ($S_{max} = 500$ in the experiment). $t_k = 1$ when the tool is picked up during the trial and 0 otherwise. Different combinations of the fitness criteria elements are tested: D, S, DS, DT, ST, S^2T and DST. Multiplication ("$\times$") is used when combining the fitness criteria elements (e.g., DS means $D \times S$)

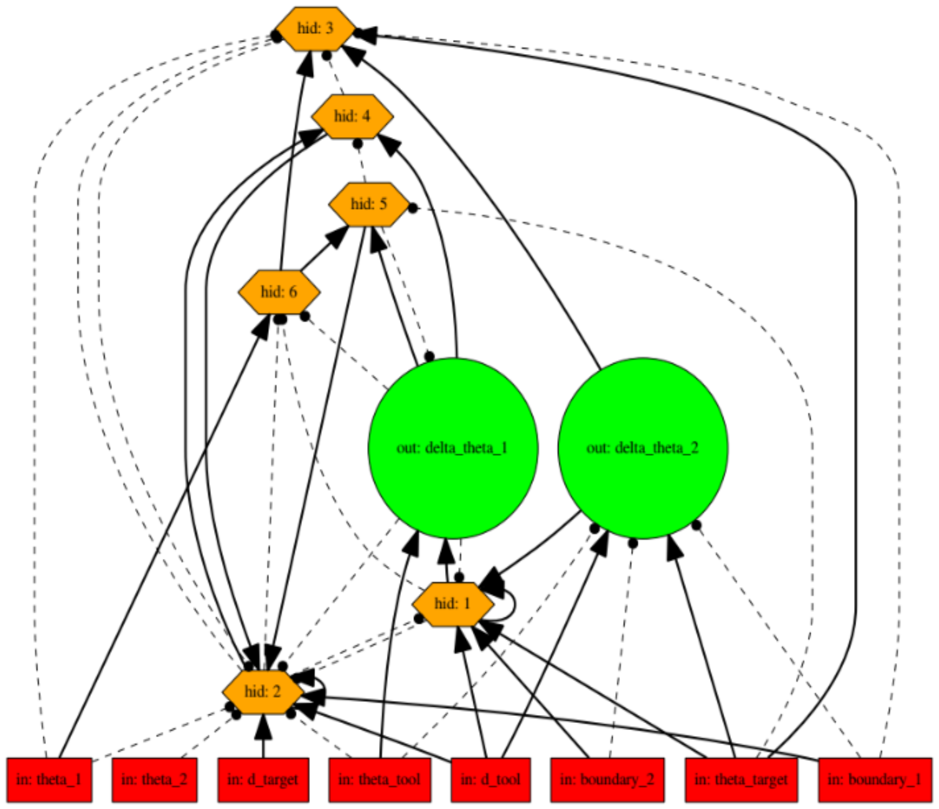


Figure 2.3: An example of the neural circuit controller. This is the connectivity graph of one the best controllers (success rate = 80.68%) in the evolution with NEAT algorithm. This controller emerged after 116 out of 120 generations. 8 input neurons (rectangles) and 2 output neurons (circles) are adapted from the original architecture of the neural circuit (before evolution) and 6 hidden neurons (hexagons) are the evolved ones. Dashed lines indicate connections with negative weights. The neuron level analysis is mainly based on this circuit because this controller has relatively simple topology compared with the other best controllers. Reprinted with permission from [3].

2.2 Experiments and Results

2.2.1 Behavior Analysis

2.2.1.1 Fitness Criteria and Success Rate

When measuring the performance of the different fitness criteria, comparing the raw directly is unfair because different fitness criteria produces different scale of fitness scores. So we simply used the success rate (the percentage of successful target reach events during a fixed number of trials) to compare the performance across different criteria. The results (mean success rate) were as follows: $D(17.78\%)$, $S(31.66\%)$, $DS(28.65\%)$ for the fitness conditions lacking T (group D, S, DS); and $DT(93.70\%)$, $ST(90.47\%)$, $SDT(94.07\%)$ for the condition with T (group DT, ST, DST). The success rates appeared similar within each group (t-test, $n = 4, p > 0.1$ for all cases) but significantly different across the two groups (t-test, $n = 4, p < 0.01$ for all case). The term T (tool pick-up frequency) turns out to be the most important feature.

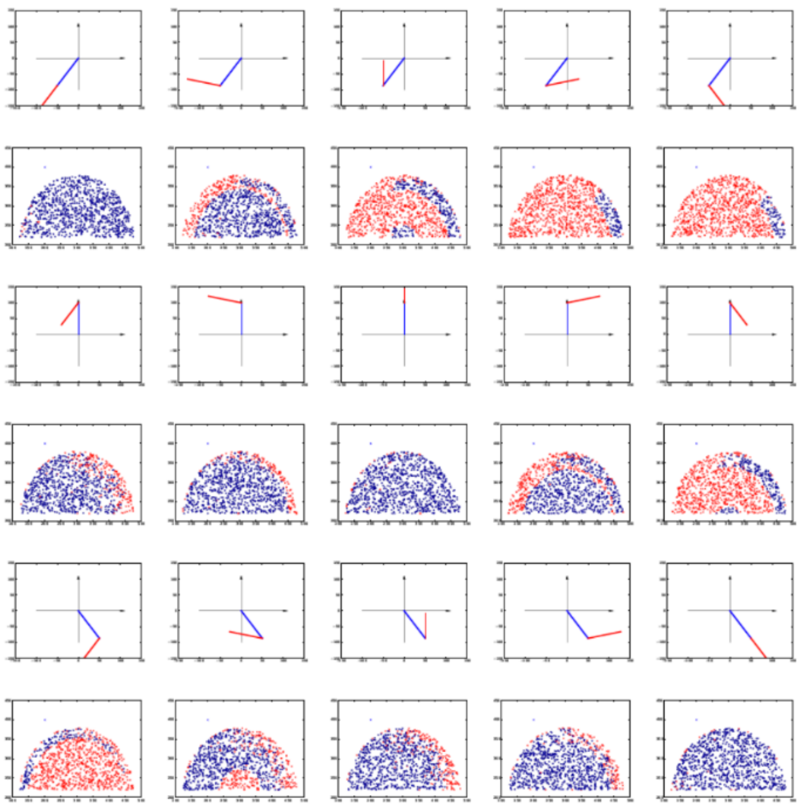


Figure 2.4: Results of evaluation with different initial trial conditions[3]. Rows 1,3,5: the starting angles of the joints. Rows 2,4,6: the corresponding random sampling of different legal tool locations (all within the reach of the arm). Each dot indicates a tool location in a trial. Dark dots indicate successful trials, and red dots indicate failed trials, respectively. The distribution changes gradually in each row as the starting angles of the arm joints change. Reprinted with permission from [3]

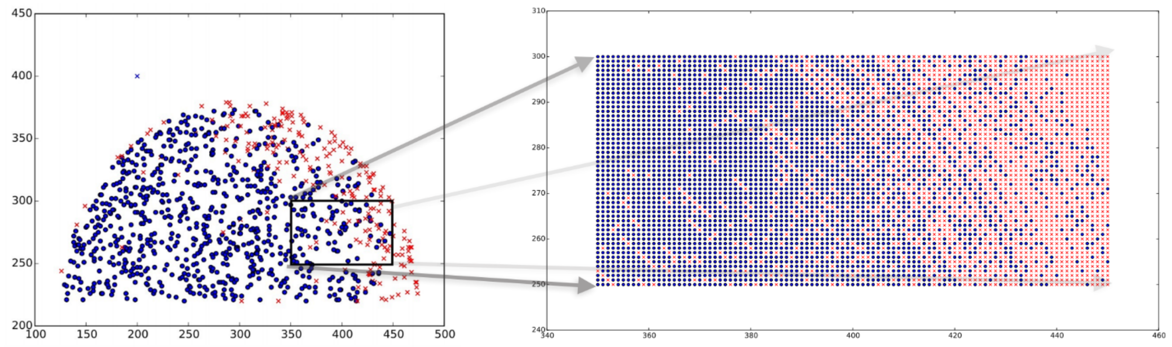


Figure 2.5: Success/fail distribution with different tool locations. Each dot indicates a tool location in a trial. Dark and red dots indicate successful and failed trials, respectively. Left: Magnified figure of the 4th row, 1st figure in Fig. 2.4. There are mixed color (mixed outcome) areas, for example, x = [350; 450]; y = [250; 300] (marked with the rectangle). Right: Enumeration of different tool locations with integer coordinates in the marked region x = [350; 450]; y = [250; 300] of the above figure. The two colors are not separable, which indicates the chaotic behavior of the arm. Reprinted with permission from [3].

2.2.1.2 Success Rate and Initial Trial Configurations

Given the initial configuration of the trial (location of the tool, target and the starting angles of the arm), the result of the trial is deterministic. We evaluated the neural circuit controller shown in Figure2.3 with different initial configurations. To investigate the correlation between the success rate and each variable, we controlled the number of variables. As shown in Figure 2.4, the correlation between success/fail distribution and starting θ_1, θ_2 of the arm joints is significant. The upper limb is represented in blue color and the lower limb is represented in red color. The initial condition is chosen from the combination of $\theta_1 \in \{-60°, 90°, 240°\}$ and $(\theta_2 - \theta_1) \in \{-120°, -60°, 0°, 60°, 120°\}$. Particularly, the success rate is the highest when $\theta_2 - \theta_1 = 0°$.

We also noticed that in Figure 2.4, there are many regions with mixed outcomes. We further investigated these regions (Figure 2.5). The arm shows some interesting behavior: very similar initial trial configurations (same target location, same starting angles of arm joints, adjacent tool locations) can lead to totally different results. The observation is that there is an approximated boundary between two regions, but the behavior of RNN controller becomes unpredictable when the target location is near the approximated boundary (see Figure 2.4). The window is generated by enumerating all possible target locations. Since there is no randomness in the RNN model itself, and the enumeration process, this result is deterministic and reproducible. In this experiment, the result is very sensitive to initial conditions. This sensitivity implies that there is possibly chaotic dynamics in the RNN model. Since the results are deterministic, the assumption here is the recurrent neural network controller has chaotic behavior.

Additional experiments on meta-configurations were done in order to verify the assumption. The meta-configurations include the angle boundary of the joint arm, the data type of the coordinates (integer vs. double precision decimals). As shown in Figure 2.6 (a)-(c), even though changing the meta-configuration can affect the results, the chaotic behavior is still preserved.

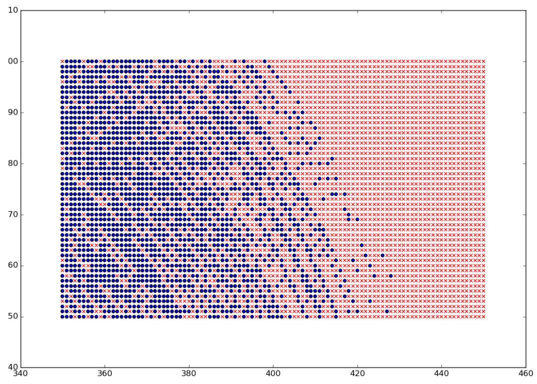
Further experiment is performed on changing the topology of the RNN controller. As shown in Figure 2.6 (d)-(f), different hidden neuron(s) are removed and then compare the test result with the original controller. The neuron(s) are selected based on the degree of connections. The least connected neuron is removed first. Though the removal of neuron can affect the result, the chaotic boundary still exists in the region.

2.2.2 Strategy Analysis

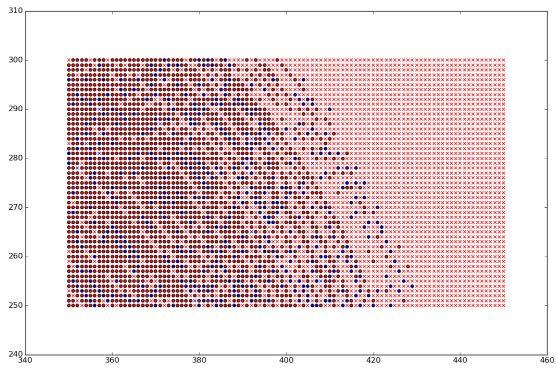
The strategy of the neural circuit controller can be evaluated from the activities of the input neurons since they contain relative agent-centered sensory representations of the environment that are changed as a result of the controller's action. We recorded the activities of 4 input neurons: relative distance and angle to the target, relative distance and angle to the tool (Figure 2.7). The convergence speed of these 4 variables can reveal the strategy of the controller.

The relative angle to the tool converge to 0 first, then the relative distance to the tool converges to 0, finally the relative angle and distance to the target converge to 0. So the strategy can be summarized as:

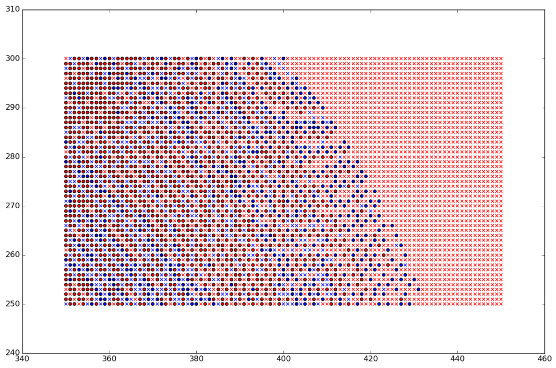
1. Minimize the relative angle to the tool,



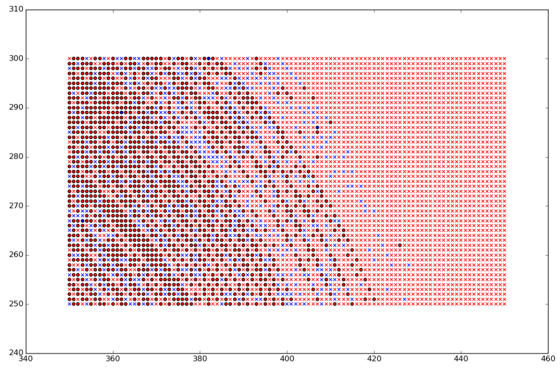
(a) Original chaotic region



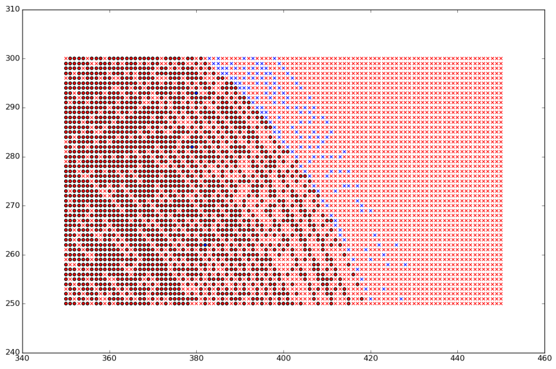
(b) No angle limit



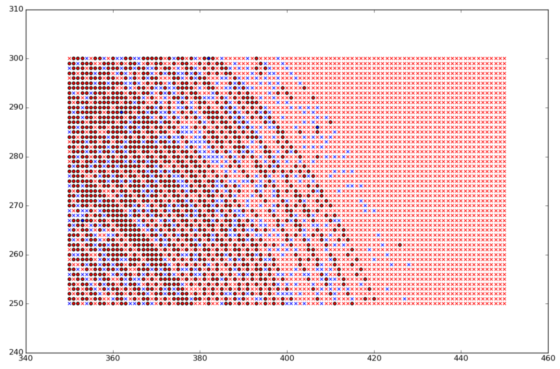
(c) Double precision



(d) Remove hid5



(e) Remove hid6



(f) Remove hid5 and hid6

Figure 2.6: (a) The original selected region. Blue dots represent successful trials and red dots represent failed trials. (b) Remove the limitation that $-60^\circ \leq \theta_1 \leq 240^\circ$. (c) Change the data type of 2-D coordinates from integer to double. (d)-(f) Change the topology of the RNN controller in Figure 2.3. In (b)-(f), blue dots represent the difference compared with the original selected region. Reprinted with permission from [3].

2. Minimize the relative distance to the tool and reach the tool,

3. Reach the target.

This strategy is also used in failed trials (Figure 2.7(b)) which indicates this strategy does not always work well. So we have to further investigate the internal state of the controller not only when the arm reaches the target, but also when the arm gets stuck and fails to reach the target.

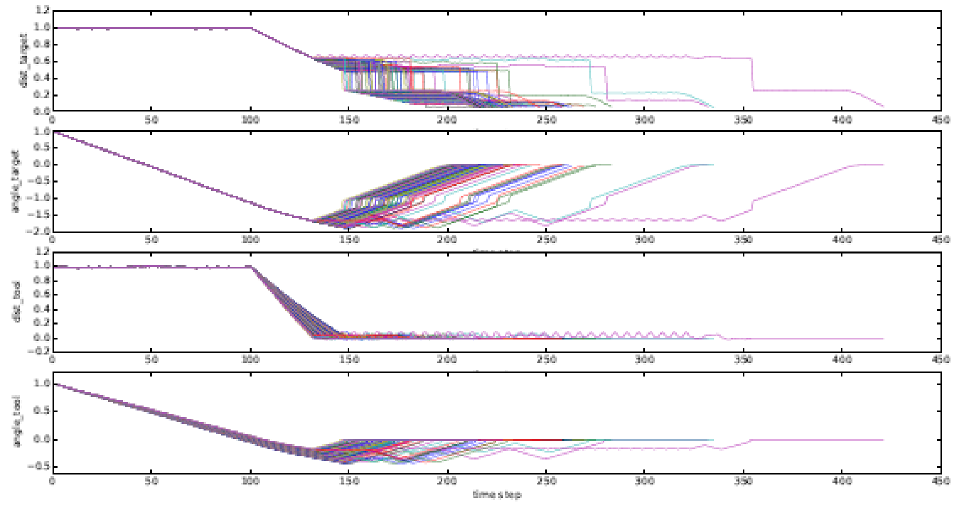
Table 2.1 shows the distribution of limit cycles in failed trials. The *Other* category may not be limit cycles since it is possible for the RNN neurons to exhibit non-periodic activity values. As shown in Figure 2.9, the neuronal activity demonstrates non-periodic behavior in some trials. 97.59% of the failed trials can be categorized as periodic orbits with period ≤ 16. The period-2 orbits (88.55%) are the dominating type of limit cycles. The trials in "other" category are either chaotic or periodic orbits with long (>16) periods (Figure 2.9). However, we cannot prove if the periodic orbits are attractors because there is still not a good method to estimate the number of the periodic orbits and their topological properties.

2.2.3 Neuronal Activity Analysis

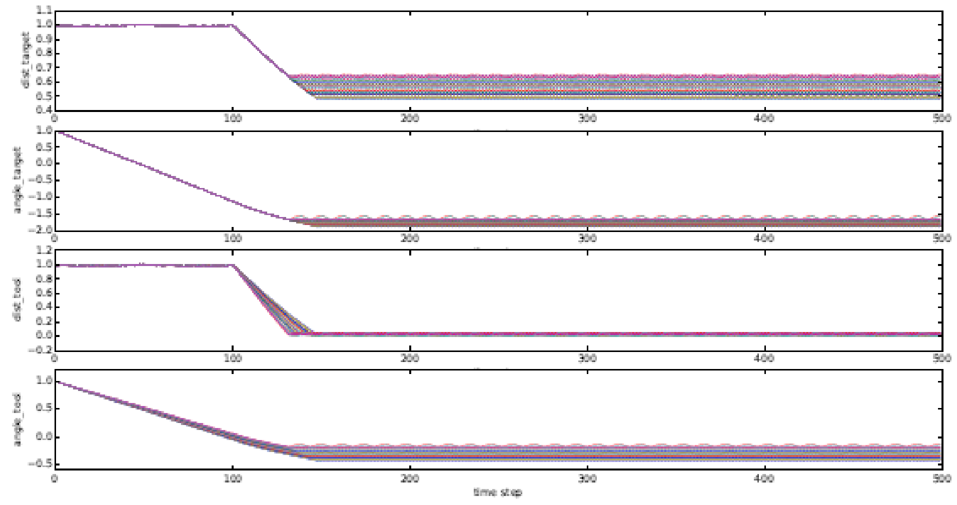
The activity of all the neurons in each trial were recorded (Figure 2.8), and we aim to find internal evidence why the arm shows such complex behavior. Since the activation function of the hidden neurons is a sigmoid function $\sigma(x) = \frac{1}{1+e^{\gamma x}}$ which is non-linear, the recurrent neural circuit can be interpreted as a non-linear dynamical system. We investigated the local stability of this dynamical system in the next section.

2.2.4 Theoretical Analysis

Even for a very low dimensional state space, the upper bound on the number of limit cycles cannot be decided. The Poincaré-Bendixson impossible theorem stressed the difficulty of the computation[55].

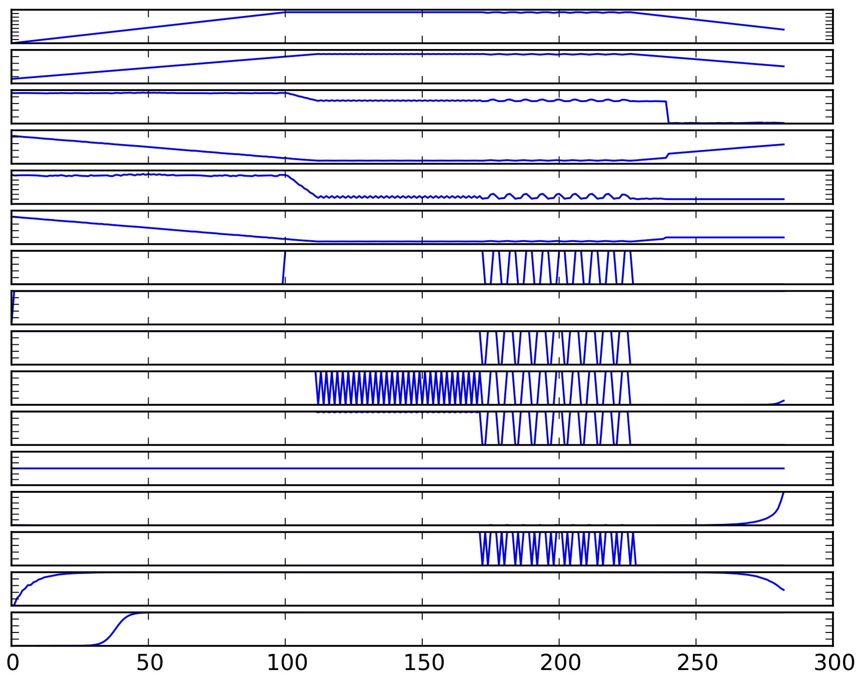


(a) Temporal activity of 4 input neurons in successful trials

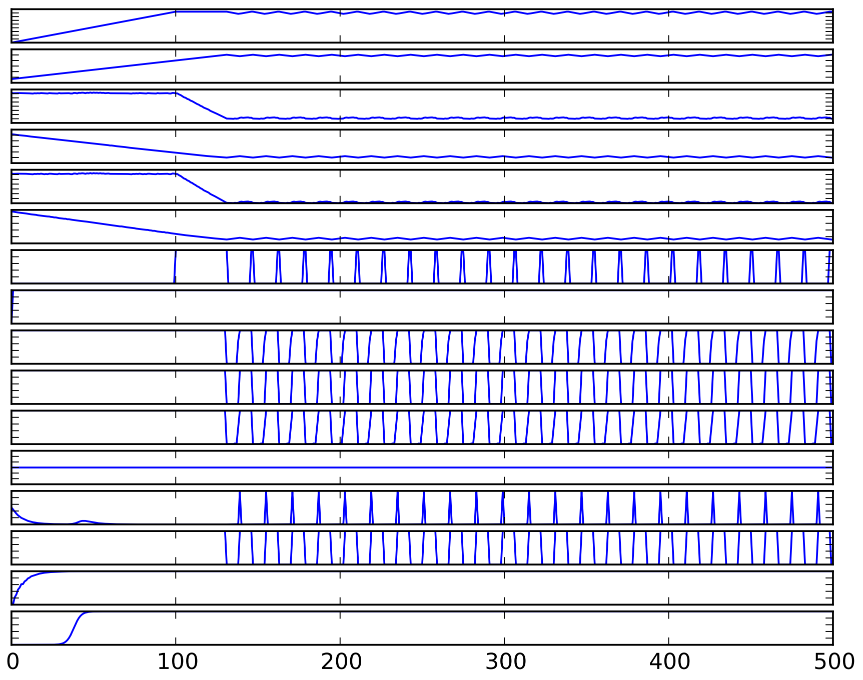


(b) Temporal activity of 4 input neurons in failed trials

Figure 2.7: Analysis of task strategy. From top to bottom: relative distance and angle to the target, relative distance and angle to the tool. The values are normalized (divided by initial value when $t = 0$). Each plot shows multiple curves from multiple trials. (a) The 4 variables have different convergence speed: relative distance to the tool is the fastest (row 3), then the relative angle to the tool (row 4), finally the relative distance to the target (row 1) and the relative angle to the target (row 2). (b) All the failed trials fail to minimize the relative angle to the tool which means they get stuck before reaching the tool, then the activities turn out to be low amplitude oscillation. Reprinted with permission from [3].



(a) Example neuronal activity of a successful trial



(b) Example neuronal activity of a failed trial

Figure 2.8: Analysis of neuronal activity. From top to bottom in each graph: 8 input neurons, 2 output neurons, and 6 hidden neurons (see Figure 2.3). (a) The trial succeeded in about 280 time steps. The sudden switch of values at about 100 time step indicates the pick-up of the tool. (b) In this failed trial, the hidden neurons started oscillation after about 120 time steps. Reprinted with permission from [3].

Type	Number of Trials	Percentage
Period-2 orbit	1693	88.55%
Period-4 orbit	102	5.33%
Period-6 orbit	23	1.20%
Period-8 orbit	11	0.58%
Period-12 orbit	23	1.20%
Period-16 orbit	14	0.73%
Other	46	2.41%
Total	1912	100.00%

Table 2.1: Distribution of limited cycles in all failed trials. Reprinted with permission from [3].

There is still no universal solution to determine the global stability of RNNs, as the Hilbert's sixteenth problem[46] remains unsolved:

> *Find a maximum natural number $H(n)$ of the number of limit cycles and relative position of limit cycles of a vector field.*

Theorem 1 (Poincaré-Bendixson Impossible Theorem). *The problem of finding a point on a limit cycle of $\dot{x} = f(x)$ in $[0,1]^2$, or the problem of determining if a given point is at most $\delta > 0$ away from a limit cycle, with black box access to a Lipschitz and continuously differentiable f, has arbitrarily high complexity.*

2.2.4.1 Solve the Fixed Points

Continuing with the analysis of autonomous RNNs, I am interested in two questions:

1. Does a fixed point exist in the state space?
2. If a fixed point exists, how is it related with the internal activities of the RNN models?

To test if there is a fixed point of the hidden neurons' activity state when the arm reaches the target, we need to know the values of the input and the output neurons. Since the location of the target is given as initial configuration, the angles of the arm joints when the arm reaches the target can be algebraically solved, therefore all the values of input neurons are known. The output

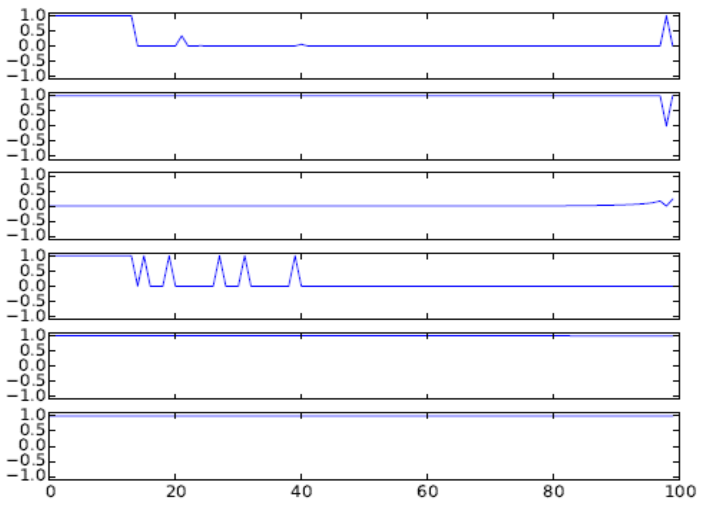


Figure 2.9: An example of chaotic hidden neuron activity in a failed trial. The activity of the last 100 time steps of 6 hidden neurons were recorded. They do not show any periodic behavior. Reprinted with permission from [3].

neurons at steady state should satisfy $\Delta\theta_1 = 0$, $\Delta\theta_2 = 0$ and can also be solved directly. Therefore, all the values of input and output neurons are known.

A fixed point holds the following condition:

$$\exists t^* \in N, \forall t \geq t^* \qquad x(t) = x(t^*) \tag{2.6}$$

Let T be a trajectory in state space. In order to solve the fixed points, we only need to solve the equation:

$$\Delta T(t) = 0 \tag{2.7}$$

Given the activation function $\sigma(x) = \frac{1}{1+e^{-\gamma x}}$, let T_i be the i-th dimension of T, then

$$T_{i(t)} = \frac{1}{1 + e^{-\sum_j \gamma w_{ji} T_j(t-1)}} \tag{2.8}$$

$$\Delta T(t) = \frac{1}{1 + e^{-\sum_j \gamma w_{ji} T_j(t-1)}} - T_i(t-1) \tag{2.9}$$

The function is differentiable, and w_{ji} and γ are constant values, thus the function is numerically solvable. The solutions are the fixed points.

This algorithm also works for activation functions other than sigmoid function. With the internal activation data of the articulated limb controller, I found that the successful trials are asymptotically close to the theoretical fixed points (see Figure 2.10).

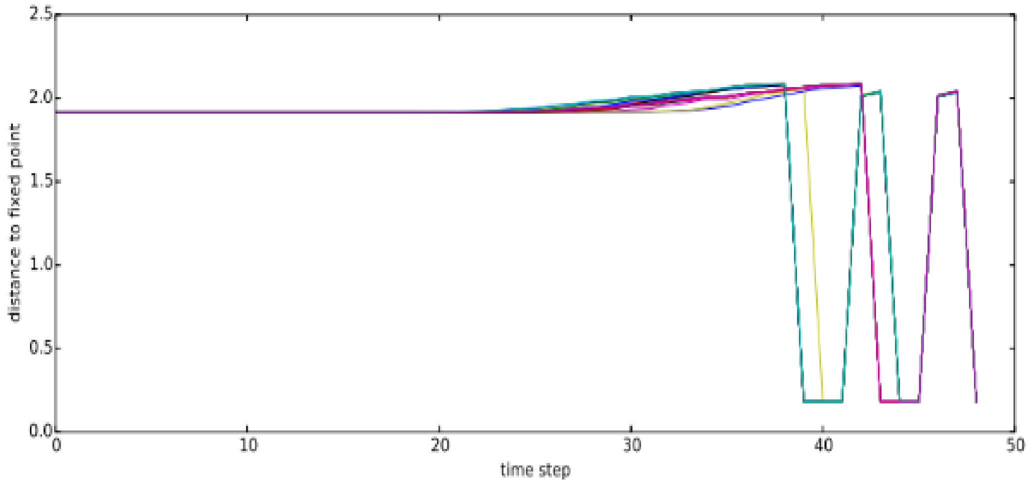


Figure 2.10: The Euclidean distances from the state vector of the hidden neurons (each curve represents a successful trial) to the theoretical fixed point. The data is captured from the last 50 time steps of successful trials in our experiment. The distance is relatively large (≥ 1.9) before the last 10 time steps, and after a short period of oscillation, the distances in all the trials decrease to very small values (≤ 0.18), indicating the end state is close to the theoretical fixed point. Reprinted with permission from [3].

2.2.4.2 *Local Stability of Autonomous RNNs*

Because the RNN model is bounded and differentiable, the local stability can be solved with the indirect approach of Lyapunov stability, based on the fact that the RNN can be locally linearized.

The Jacobian matrix at fixed point $T(t)$ is

$$J = \left[j_{ij} \right] = \left[\frac{\partial T_i(t)}{\partial T_j(t-1)} \right] = \left[w_{ji} T_i(t)(1 - T_i(t)) \right] \tag{2.10}$$

The eigenvalues of J are the indicators of local stability at $T(t)$. The Jacobian matrix is sparse since the network is not fully connected. In the successful trials of our experiment, all the eigenvalues of the Jacobian matrix with the solution of Equation 2.2.4.2 turn out to be real and negative, which indicates that the fixed points are stable. The fixed point is also an attractor for all the successful trials (Figure 2.10).

Theorem 2 (Lyapunov Stability Theorem)**.** *The fixed point $T(t)$ is asymptotically stable if and only if all eigenvalues have negative real parts.The fixed point $T(t)$ is unstable if one or more of the eigenvalues has positive real part.*

Theorem 2 [56] can be used to judge the local stability, though it does not tell anything about region of attraction. The fixed point in Figure 2.10 is proved to be asymptotically stable by this theorem.

An example of the Jacobian matrix is (using the fixed point p in Figure 2.10):

$$p = \begin{pmatrix} 1.0000 \\ 0.0000 \\ 0.0043 \\ 0.8199 \\ 0.9995 \\ 0.0098 \end{pmatrix} \tag{2.11}$$

$$J = \begin{pmatrix} -1. & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0.0007 & 0 & 0 \\ 0 & 0 & 0 & -1 & -0.5516 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ -0.0022 & 0 & 0 & 0 & 0 & -1 \end{pmatrix} \tag{2.12}$$

The eigenvalues are $[-1, -1, -1, -1, -1, -1]$.

2.3 Contribution and Discussion

In this chapter, we analyzed the behavior and the neuronal activation of the autonomous recurrent neural network which was used in the task of controlling an articulated limb for tool use, and showed the correlation between the behavior of the limb and the internal dynamics of the RNN. We also investigated the local stability of the end states in both successful (goal state reached) and failed trials in our experiment. The existence of fixed points and limit cycles could explain the cause of success and failure. We expect our results to help better understand RNNs and their dynamics.

At present, not only in our previous experiment, but also in many complex control problems using neuroevolution [28], fitness criteria and success rate estimation are mainly done by simple random sampling of initial task conditions. However, simple random sampling often ignores the variation in behaviors and internal dynamics across different sub-populations. Stratified sampling we used in this paper can be more helpful in finding the difference of behavior among sub-populations across trials (Figure 2.5). The overall task performance can be evaluated from the performance of many sub-tasks, but not vice versa.

Though global stability analysis is not practical for a high-dimensional non-linear dynamical system [28] like the one in our experiment, to some extent, local dynamical analysis can still explain why and how a trial can succeed or fail. The target in the tool-use task puts an attractor in the state space of hidden neurons. However, some trials may step into limit cycles so that the arm fails to reach the target. We hope our study can provide new performance criteria (e.g. the comparison of performance among sub-populations with different configurations, the stability of the goal state, etc.) in the evaluation of evolved RNNs.

The limitations of the study in this paper are as follows: most world properties were heavily encoded in the inputs themselves, e.g., target location, tool location, etc; our method still cannot predict when will the state of hidden neurons get trapped in limit cycles. To improve the neural circuit controller, object recognition and grounding [57][58], and efficient codes for motor encoding [59] can be incorporated into our system for increased realism. We can also integrate our behavior analysis with novelty search [60]. Since the behavior analysis is independent of the fitness function, it is possible to systematically build up the behavior space based on behavior analysis. More extensive lesion study [61] on the internal dynamics of the RNNs can also be used. This will help understand thoroughly the functionality of each component in the network and to improve the robustness of the RNNs.

3. ANALYSIS OF NON-AUTONOMOUS RECURRENT NEURAL NETWORKS

The hidden states of non-autonomous recurrent neural networks depend on both the previous hidden state and the current input: $h_t = f(h_{t-1}, x_t)$. When x_t and h_{t-1} are independent with each other, the recurrent neural network is non-autonomous. Non-autonomous recurrent neural networks refer to the category of recurrent neural networks which take partially observable data as input to the neural network. The non-stationary time series analysis [62] tasks require the models to be non-autonomous, such as weather forecast [63][64], speech recognition [43][65], hand-writing recognition [66][67] and machine translation [25][68][17].

The non-autonomous recurrent neural networks can be represented as non-autonomous dynamical systems. The dynamics of non-autonomous dynamical systems take place on smooth fiber bundles[69] [70]. The state space of a non-autonomous dynamical systems evolves over time and a fiber bundle contains continuous surjective map between state spaces at different time steps. As shown in Figure 3.1, A point in the fiber $q = (x, g) \in Q = M \times G$ projects down to the point $x \in M$ under the projection map $\pi : Q \mapsto M$ defining the principal fiber bundle [71]. Unlike autonomous systems, the non-autonomous systems are difficult to analyze mathematically. The local stability usually does not sustain with external inputs.

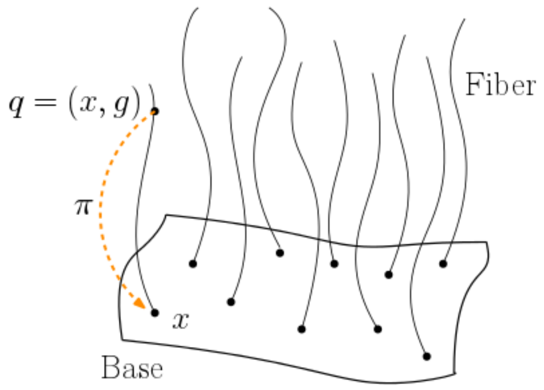


Figure 3.1: Schematic of a fiber bundle. Illustration adapted from [4].

Despite the conventional methods for time series analysis such as regression and hidden Markov models are highly interpretable [72], these methods are usually based on some strong assumptions. For example, regression models assumes that the data to follow some particular distribution (uniform, Gaussian, etc.), and the observations are independent with each other[73]. Hidden Markov models assume that the input to have Markovian property [74]. These assumptions create difficulties for the models in achieving state-of-the-art performance on complicated tasks.

Recurrent neural networks do not rely on such assumptions, and have been a powerful tool for various time series analysis tasks. However, the dynamics of recurrent neural networks is hardly known. To understand the internal dynamics is of vital importance for analyzing the robustness and reliability of RNNs. This chapter includes my work on analyzing the internal dynamics of LSTM models with different tasks.

3.1 Task Description

The goal of this study is to investigate the dynamics of RNN models that take external inputs. As shown in Figure 3.2, the RNN classifiers are built with LSTM cells [24] and GRU cells [25] since they are good models for memorizing long term dependency in the input sequence. The output vector from LSTM cells are fed into a dense layer and the final output vector is softmaxed. A softmax vector has normalized property, so that it can be used in categorical representation.

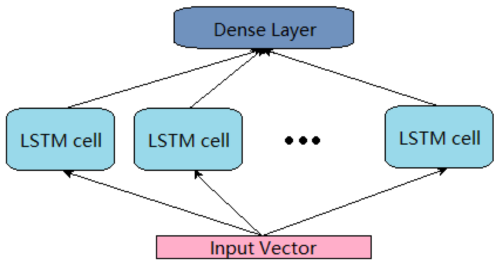


Figure 3.2: The configuration of RNN model with LSTM cells. Various number of LSTM cells are tested.

3.1.1 Sequential MNIST Classification Task

MNIST is a well-known handwritten digits database. Each image in the database has $28 \times 28 = 784$ pixels and there are 10 classes for each digit from 0 to 9. The data set contains 60,000 training samples and 10,000 testing samples.

Sequential MNIST classification is to reshape the image into a sequence of pixels or a sequence of pixel vectors as input, and to classify the sequential input. In my experiment, the 784 pixels are converted to 28 vectors with 28 pixels in each vector. In other words, the RNN classifier scans the image row by row, as shown in Figure 3.3.

To recognize the digits, the RNN classifier has to keep an internal memory of the past input, since a single row does not provide sufficient information for the digit identification.

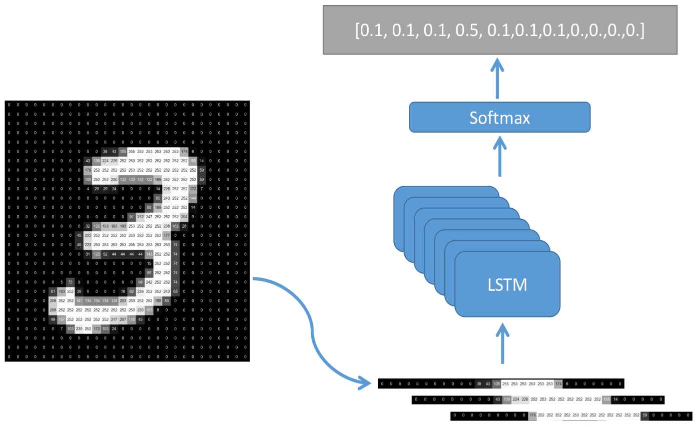


Figure 3.3: Illustration of the sequential MNIST task

3.1.2 DNA Sequence Classification Task

Splice junctions [75] are points on a DNA sequence at which "superfluous" DNA is removed during the process of protein creation in higher organisms. The data set was acquired from the UCI machine learning repository [76]. The problem posed in this data set is to recognize, given a sequence of DNA, the boundaries between exons (the parts of the DNA sequence retained after splicing) and introns (the parts of the DNA sequence that are spliced out). This problem consists of two sub-tasks: recognizing exon-intron boundaries (referred to as EI sites), and recognizing intronexon boundaries (IE sites). (In the biological community, IE borders are referred to a “acceptors” while EI borders are referred to as "donors".) The DNA sequence data set contains 3,000 samples, and is randomly shuffled into training and validation data set with a 4:1 ratio. A DNA sequence example looks like

CCAGCTGCATCACAGGAGGCCAGCGAGCAGGTCTGTTCCAAGGGCCTTCGAGCCAGTCTG

As shown in Figure 3.4, each DNA sequence has 60 base pairs, and 4 base types: A,T,G and C. The RNN model is trained with input of 4 dimensions$\times$ 60 time steps. Similar to the sequential MNIST classification task, recurrent (LSTM/GRU) cells are used in the RNN classifier.

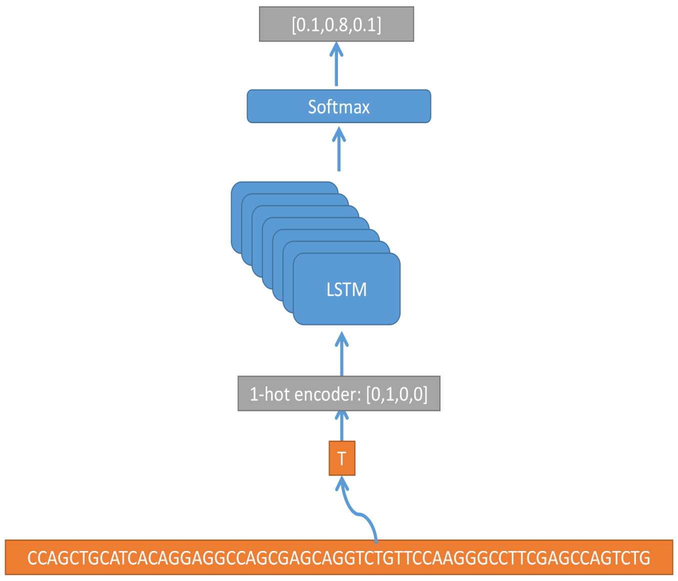

[0.1,0.8,0.1]
Softmax
LSTM
1-hot encoder: [0,1,0,0]
T
CCAGCTGCATCACAGGAGGCCAGCGAGCAGGTCTGTTCCAAGGGCCTTCGAGCCAGTCTG

Figure 3.4: Illustration of the DNA classification task.

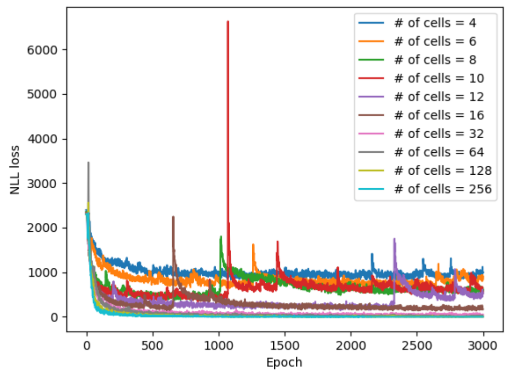
3.2 Experiments and Results

3.2.1 Training

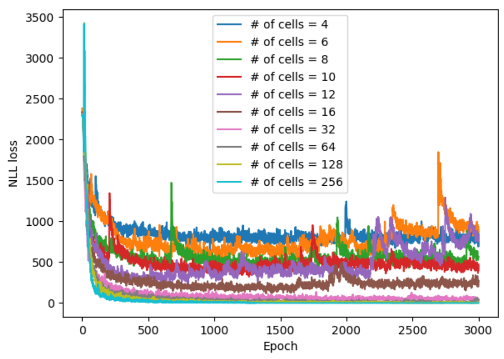
Implemented with PyTorch, the RNN models were trained and tested on Intel i7-6700K CPU and a single NVidia GTX1070 GPU. The first step is to evaluate the performance of RNN classifiers. Models with different numbers of cells were tested, and since dropout [77] directly affects the number of cells used in the training phase, the results of using dropout and without dropout are also compared (see Table 3.1, Table 3.2 and Figure 3.6). 30 models were trained with the same training data set for each number of cells per task, and the accuracy is averaged among all models with the same configurations. The LSTM and GRU models are trained with two configurations with dropout=0.5, and dropout=0 as baselines. The effect of dropout on robustness will be discussed in the follow-on parts. The RNN models were optimized with standard stochastic gradient descent (SGD) [78] algorithm on mini-batches. The negative log likelihood (NLL) loss function [79] curves can be found in Figure 3.5. The NLL loss is equivalant to cross entropy as the cross entropy loss can be represented in

$$loss(x, class) = -\log \frac{\exp(x[class])}{\sum_j \exp(x[j])} = -x[class] + \log \sum_j \exp(x[j]) \tag{3.1}$$

The training part is straightforward and after training, the RNN models can eventually reach certain performance, which can be used for further experiments.



(a) Loss curves of LSTM models with dropout=0.



(b) Loss curves of LSTM models with dropout=0.5.

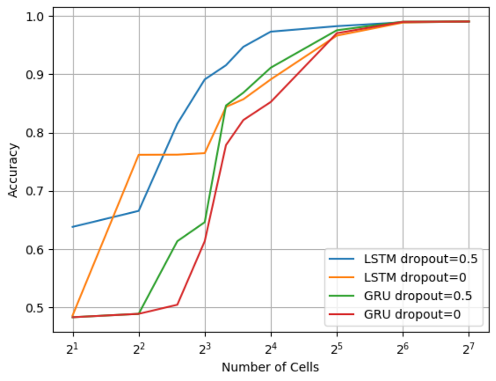
Figure 3.5: Loss function curves for different LSTM models. Use of mini-batches leads to the sudden peaks in the curves.

# of Cells	LSTM(dropout 0.5)	LSTM	GRU(dropout 0.5)	GRU
4	63.82%	48.64%	48.32%	48.32%
6	66.56%	76.19%	48.90%	48.90%
8	81.51%	76.21%	61.36%	50.46%
10	89.11%	76.46%	64.62%	61.36%
12	91.55%	84.40%	84.64%	77.86%
16	94.71%	85.71%	86.84%	82.15%
32	97.31%	89.12%	91.13%	85.25%
64	98.25%	96.63%	97.55%	97.04%
128	98.92%	98.87%	99.02%	99.00%
256	99.01%	99.06%	99.06%	99.03%

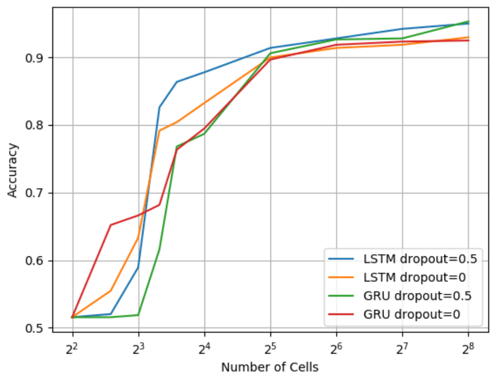
Table 3.1: Sequential MNIST Classification accuracy of different models.

# of Cells	LSTM(dropout 0.5)	LSTM	GRU(dropout 0.5)	GRU
4	51.57%	51.57%	51.57%	51.57%
6	52.04%	55.49%	65.20%	51.57%
8	58.93%	63.32%	66.61%	51.88%
10	82.60%	79.15%	68.18%	61.60%
12	86.36%	80.41%	76.33%	76.68%
16	87.77%	83.23%	79.47%	76.80%
32	91.38%	89.97%	89.66%	90.60%
64	92.79%	91.38%	91.85%	92.63%
128	94.20%	91.85%	92.32%	92.79%
256	94.98%	92.95%	92.48%	95.30%

Table 3.2: DNA Classification accuracy of different models.



(a) Curves of accuracy in sequenctial MNIST task.



(b) Curves of accuracy in DNA classification task.

Figure 3.6: Accuracy vs. number of cells for different LSTM models.

3.2.2 Experiment on Recurrent Cell Dynamics

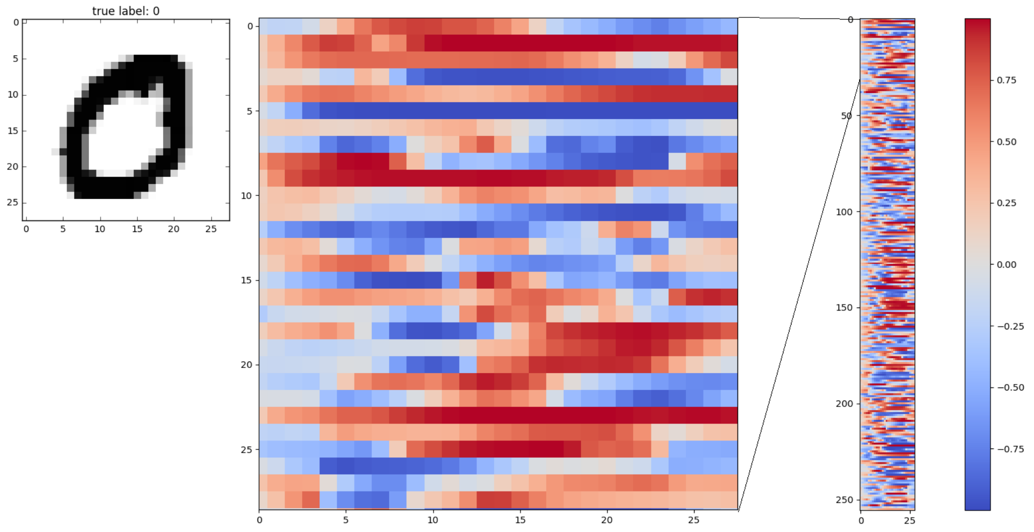


Figure 3.7: The hidden activation h_t of a 256-cell LSTM neural network on an MNIST test sample. X-axis represents the time step. The colors represent the values of h_t.

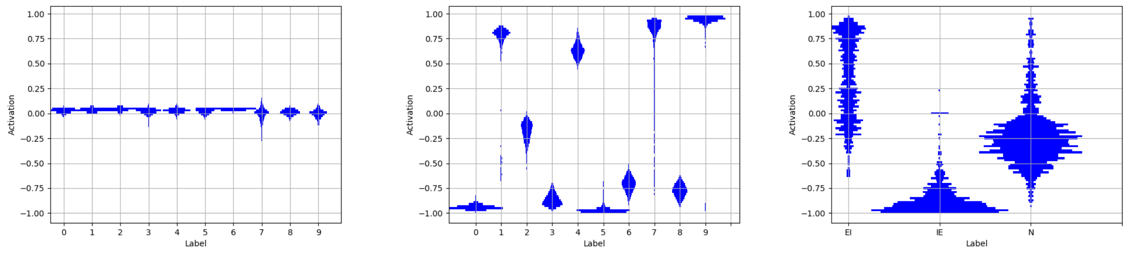
Figure 3.7 shows the hidden state h_t histogram of a LSTM cell for sequential MNIST task. Different cells exhibit different dynamics on the same input. As shown in Figure 3.8 (b), statistically, a well-trained LSTM cell is able to learn the differences across different classes. We observed that similar distributions between digit 3,6,8 and between digit 1 and 7. And we also observed significant deviation in the digit 7 histogram in Figure 3.8(b). This result can be explained with the spatial features from the digit images, which are typically learned in convolutional neural networks. However, the LSTM also demonstrates such characteristic in the experiment based on this result.

This result intuitively makes sense because the handwritten digits 3, 6 and 8 usually share common parts, and so are 1 and 7. However, the way of writing digit 7 varies as some people put a stroke across the digit, as shown in Figure 3.9.

In Figure 3.8(c), the divergence of histograms across different classes is also significant. For any of the class pairs, the divergence can be quantified with Kullback-Leibler divergence (KL-divergence) [80], which is also called relative entropy. Suppose P and Q are two discrete probability distribution, the KL-divergence can be represented as:

$$D_{KL}(P||Q) = \sum_{x \in \chi} P(x) \log \frac{P(x)}{Q(x)} \tag{3.2}$$

The increase in the relative entropy implies information gained through training.



(a) h_t of an LSTM cell before training for sequential MNIST task.

(b) h_t of an LSTM cell after training for sequential MNIST task.

(c) h_t of an LSTM cell after training for DNA classification task.

Figure 3.8: Example hidden state h_t histograms for untrained and trained LSTM cells with all training samples. X-axis represents the digit labels from 0 to 9 in (a)(b), and EI, IE, N in (c).

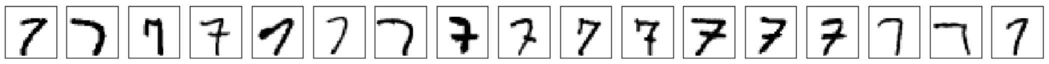


Figure 3.9: Examples of handwritten digit 7 in MNIST data set. The stroke in the middle causes larger variance in pixels than the other digits.

In the DNA experiment we also want to investigate the trajectories of the sequences in the state space (see Figure 3.10). The linear layer that connects the hidden states of recurrent cells and the output vectors is a surjection so that we can first investigate the trajectories in the output state space. The DNA has 60 components and 3 classes so that there is no need to apply dimension reduction embedding for visualization. Since the convergence for an non-autonomous dynamical system can not be derived directly, we extend the input sequence with zero padding to simulate the resting state of the RNN model, as from T=60 to T=200 in Figure 3.10. With idle input, the RNN model eventually works as an autonomous dynamical system.

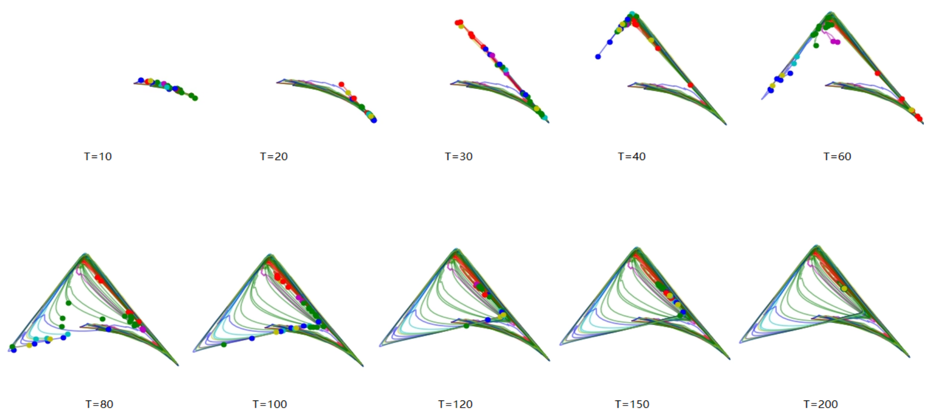


Figure 3.10: Example of trajectories of output vectors in DNA classification task, from $t = 10$ to $t = 200$. In this case, all trajectories converge to one point by the end.

Interestingly, the RNN models do not always converge with zero input which implies the RNN model is not always input-to-state stable [81] (see Figure 3.11). Another hypothesis is that the larger the number of LSTM cells is, the more likely the state space becomes unstable.

3.2.3 Latent Feature Space Exploration

This experiment aims to investigate the latent feature space via the hidden states of the recurrent units from the sequential MNIST task. As shown in Figure 3.7, the hidden states demonstrate various dynamics. Intuitively, we want to know more about the trajectories of the hidden activities in the latent space. In order to analyze the high-dimensional latent space, we applied some

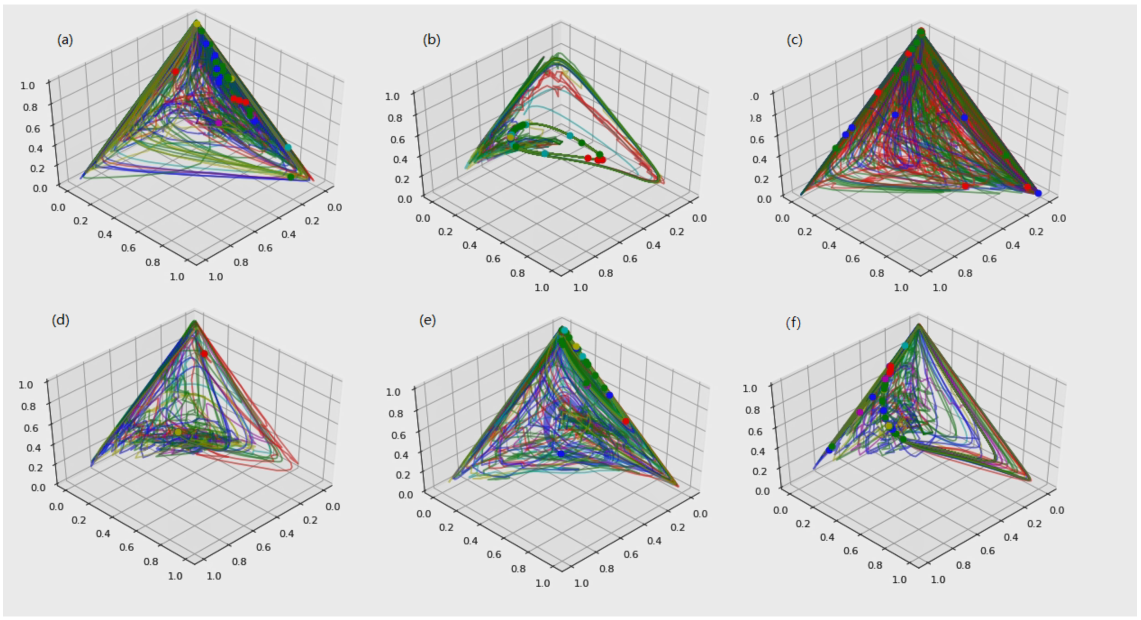


Figure 3.11: Trajectories of test samples from different RNN models in latent space. a) both attractor and limited cycle exist in state space, b) limit cycle exists in state space, c) chaotic trajectories, d) more than 1 attractors exist in state space, e) more than 1 limited cycles exist in state space, f) periodic but strange trajectories.

dimension reduction methods such as t-SNE [82], Principal Component Analysis (PCA) [83] and auto-encoder [84][12]. In this experiment, the auto-encoder is a fully connected neural network that encodes a hidden state vector into an encoded 2-D vector and decodes the 2-D vector to the hidden state vector itself, and the encoded vectors are used in 2-D visualization. As shown in Figure 3.13, the feature representations have good convergence on the image data. The PCA and auto-encoder embeddings demonstrate similar characteristics (see the supplemental plots in Appendix).

As can be seen in Figure 3.12, when projected to a 2-D plane with t-SNE, the hand-written image samples do not show clear boundaries between classes. In contrast, as shown in Figure 3.13, the ending points of the trajectories demonstrate loose boundaries between classes. The trajectories depict the relative distance between different samples on each time step. The samples start with blank pixels, so that the trajectories start near the same neighbourhood. The divergence happens

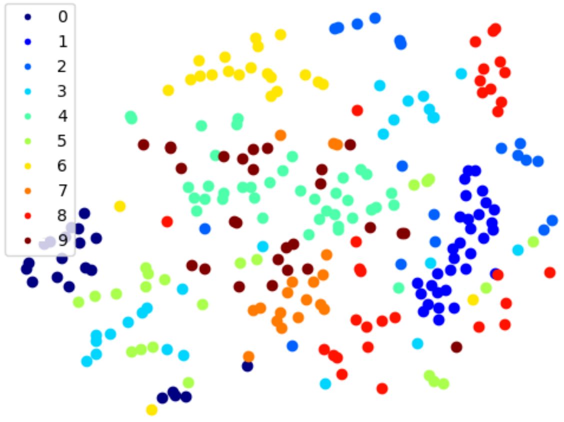


Figure 3.12: 2-D t-SNE of a subset of MNIST input images.

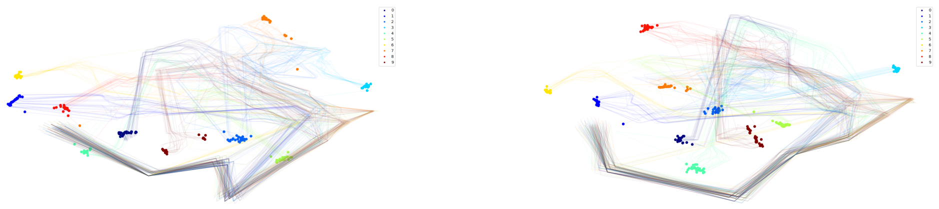


Figure 3.13: 2-D t-SNE of the hidden states in a 256-cell LSTM classifier, with different subset of MNIST images. Full-size image can be found in the Appendix.

when the input vectors start to have variations. Although variations between two consecutive time steps within the same class is also non-trivial, the trajectories of the same class do not converge as much as different classes.

3.2.4 Experiment on Sequential Representation

The recurrent neural networks have response on each step of the sequential input. Although in the accuracy table 3.1 we only measure the result from the last step, we are also curious about the intermediate responses along each step of the sequence, and how the model complexity affects feature extraction.

Figure 3.14 demonstrates the temporal change of the sequential representation at the softmax layer. The predicted label can be calculated as:

$$L(x) = \underset{p \in P}{\operatorname{argmax}} \frac{e^{p(x)}}{\sum_{p \in P} e^{p(x)}} \tag{3.3}$$

in which $p \in P$ denotes the linear combination of h_t in recurrent layer. In Figure 3.14, if we compare different models, we find that larger RNN models provide higher confidence level in terms of the softmax output than smaller RNN models. Recall the result from previous experiment (see Figure 3.8), the trained recurrent cells gain higher relative entropy in training. As a result, at similar level of cross-entropy convergence, the linear combinations of h_t in larger models are more diverged, so that the softmax outputs have much higher bias than smaller RNN models.

Another observation is that a sub-sequence from the whole input can be sufficient for making correct predictions. This is an effect caused by backpropagation through time (BPTT) in the LSTM/GRU training. With the given sub-sequence of input, the RNN models are able to forecast the upcoming sequence.

We also observe that the softmax vectors do not change smoothly with time. Consider the surjection from h_t to the softmax vectors, this phenomenon reflects the bifurcations of trajectories in the hidden state space, as shown in Figure 3.13.

To further confirm the above analysis on the empirical result, we tested the RNN models with mutated input (see Figure 3.15). The small RNN models are very sensitive to mutations however the large RNN models are not. The partially erased sub-sequence still works well in this experiment, and the non-smoothness also increase with the complexity of the RNN models.

3.2.5 Experiment on Neural Network Lesion Study

Another experiment aims to investigate the robustness of the trained RNNs. The lesion is applied by randomly removing the connections between recurrent cells and the softmax layer.

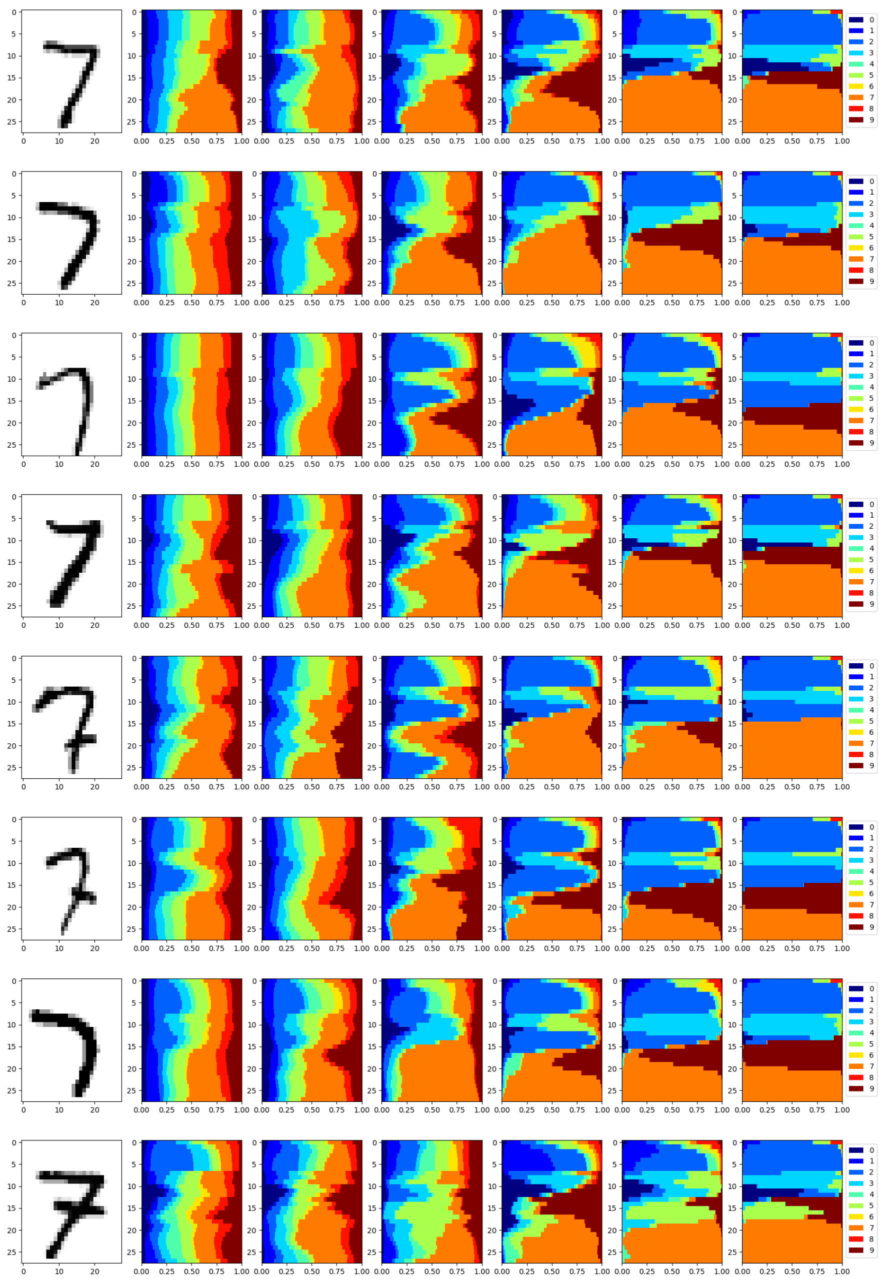


Figure 3.14: Examples of the input images and the corresponding sequential representations. Starting from step 0 to step 27, the colored stacked histograms demonstrate the changes in softmaxed prediction vectors. From left to right, each column corresponds to the trained model with 8, 16, 32, 64, 128, 256 LSTM cells, respectively. The x-axis in the histogram represent the values in the softmax output vector. Additional examples can be found in Figure A.4.

We gradually increase the percentage of the number of disconnected cells, from 0 to 90% and the result is shown in figure 3.16. The experiment is repeated 10 times to achieve better estimation of the decay in accuracy. The result indicates that with lesion applied, the accuracy decreases close to the accuracy from a trained model with equivalent number of cells, which can be fitted with the same curve.

We also observe that dropout while training improves the robustness of the RNNs against lesion due to its enhancement to the functionality of a sub-network during training.

3.3 Contribution and Discussion

The trajectory study reveals that even for simple recurrent structure, given external output, the internal dynamics can be quite complicated, and a model that achieves good performance can still be unstable. This experiment can be further extended in many aspects. e.g., how the chaotic behaviors affect the whole model's performance needs to be investigated, and how perturbed input affects the stability of the model, and finally, how to translate the trajectories into classification strategies.

The sequence representation study shows how the intermediate output is related with the input sub-sequence. It is worth noticing that partial data may be sufficient for inference, and this may provide some insight for compressed sensing [85]. The comparison across models with different complexity provides new methodology for understanding the model generalization [86] in machine learning tasks.

The lesion study demonstrates that LSTM/GRU networks are robust against connection removal from fully-connected layers. Although the performance decays with lesion, it is still comparable with trained models that has equivalent number of cells. Based on these results, the recurrent neural networks can be further optimized to appropriate number of cells.

Nevertheless, some additional work can be done on the RNN models with partial input. Investigation on the trajectories in the latent space would help us understand how RNN models make decisions.

The effectiveness of the dropout in the training phase can also be integrated with information theory. My hypothesis is that dropout will add bias to the information gain across RNN cells, and as a result, the redundancy of the RNN is enhanced by dropout.

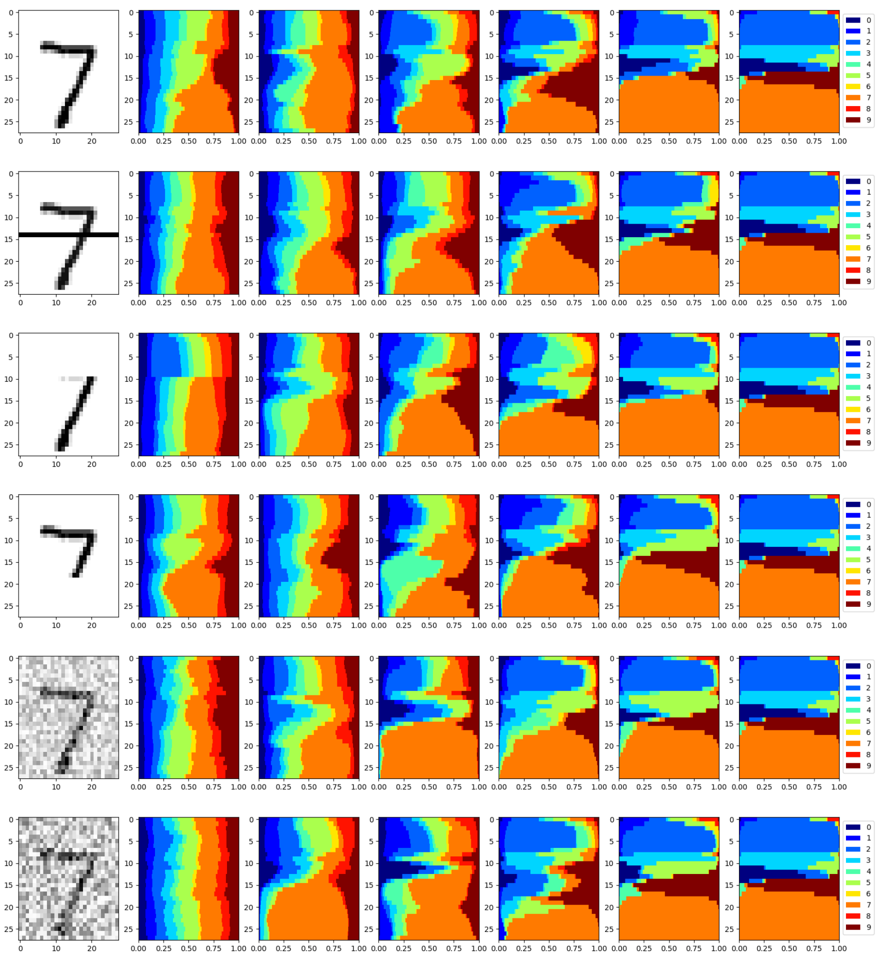


Figure 3.15: Examples of the mutated input images and the corresponding sequential representations. Starting from step 0 to step 27, the colored stacked histograms demonstrate the changes in softmax-ed prediction vectors. From left to right, each column corresponds to the trained model with 8, 16, 32, 64, 128, 256 LSTM cells, respectively. From top to bottom, each row corresponds to (1) original input, (2) with stroke in the middle, (3) with first 10 steps erased, (4) with last 10 steps erased, (5) Gaussian noise with PSNR=10dB (6) Gaussian noise with PSNR=5dB, respectively.

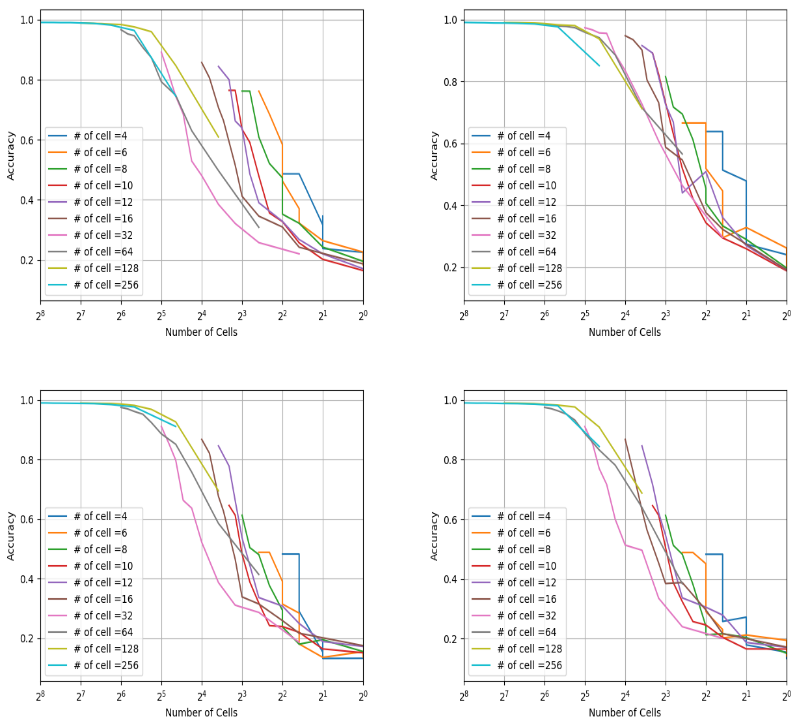


Figure 3.16: Lesion test on Sequential MNIST task. Top-left: LSTM, top-right: LSTM with dropout=0.5, bottom-left: GRU, bottom-right: GRU with dropout=0.5. Each curve represents a model and x-axis represents the number of remaining cells.

4. RECURRENT NEURAL NETWORK COMPRESSION

4.1 Motivation and Task Description

Based on the redundancy and robustness observation from section 3, it is highly possible to compress LSTM networks by reducing the number of parameters without much loss in performance [87]. The idea of neural network compression could date back to 1990 [88]. However, the demand for neural network compression had not been significant until deep learning [11] became popular. The neural networks compression does not only accelerates the computation, but also increases the hardware consumption efficiency. Many works have been done on the convolutional neural network (CNN) compression [20][89][90][91]. However, it is yet unknown whether these methods are applicable to recurrent neural network compression. Unlike feed-forward neural networks in which the number of parameters are generally linearly proportional to the number of hidden units, an RNN has hidden-to-hidden transition matrix in which the number of parameters is quadratic to the number of hidden units. As the hidden structure grows, the computational complexity significantly increases. Therefore, complexity reduction methods are needed to enable larger models running on less powerful devices such as cell phones and smart watches.

One way to reduce the computational complexity is to decrease the floating point precision in neural network parameters [92]. Another way is to utilize knowledge transfer to train a smaller model to mimic larger model [93]. Some other methods utilize the trick of sharing weight sharing [94][20]. However, the majority of CNN compression tasks are performed via pruning. In the pruning algorithms, the convolutional kernels and fully-connected nodes are measured by some importance criteria, and the least important convolutional kernels are removed from the network. Usually a re-train process is need after pruning, and this pipeline can be applied iteratively [90].

The task in this chapter is to propose and verify an efficient algorithm for RNN compression. The experiments were on top of Section 3 for comparison with the baseline results and implemented in PyTorch.

4.2 Proposed Method and Results

Since RNNs demonstrate robustness in the lesion test, and the LSTM cells on the same layer are independent with each other in terms of cell output, pruning is a feasible approach to achieve the goal of reduction in number of parameters without significant performance drop. The LSTM network can be pruned at the level of LSTM cells.

Inspired by the pruning algorithms for CNN [89][20], I propose an algorithm to prune the LSTM cells by removing the connections between the cells and the adjacent fully-connected layers. The LSTM cells are ranked by the sum of the absolute weights $\Sigma|F_{i,j}|$, i.e. its l_1-norm $\|\mathcal{F}_{i,j}\|_1$. Based on the assumption that better cells/neurons have larger absolute weights, the l_1-norm represents the expected capacity of the LSTM cell to contribute to the output feature map. Although this assumption can be observed empirically (see Figure 4.1), since training neural networks is a non-convex optimization problem, mathematical proof seems still infeasible. Compared with other known pruning algorithms [90][92], l_1-norm calculation does not rely on data, and has very low computational complexity. Some other criteria such as Wasserstein distance [95] and Hellinger distance [96] were also examined, but they are much less efficient and did not provide better performance on general tasks.

Here I propose two algorithms: one-shot and iterative RNN pruning (see Algorithm 1 and 2). In one-shot pruning algorithm, only the desired number of cells n' needs to be decided. Based on the importance of cells, only the most important n' cells are kept. In iterative pruning algorithm, every iteration eliminates k least important cells before retraining the model until the network reduces to the desired number of cells n'.

Figure 4.2 shows the RNN pruning result. 256-cell LSTM has only 0.39% accuracy loss with 224(87.5%) cells being pruned. 256-cell LSTM with dropout has 0.66% accuracy loss with 87.5% cells being pruned. LSTM without dropout shows higher compression capability than LSTM with dropout (see Figure 4.4). Figure 4.3 shows the result of algorithm 2. The two algorithms reach similar compression results.

Figure 4.4 and Table 4.1 demonstrate the performance of pruning algorithm on sequential

Algorithm 1 One-Shot RNN Pruning

```
1: $W \leftarrow$ trained dense layer weight matrix
2: $n' \leftarrow$ desired number of cells
3: $n \leftarrow$ current number of cells
4: $\vec{F} = \Sigma_j |F_{i,j}|$
5: $L = argsort(\vec{F})$
6: for $i = 1...n - n'$ do
7:     $\vec{W_i} = 0$
8: re-train the neural network
```

Algorithm 2 Iterative RNN Pruning

```
1: $W \leftarrow$ trained dense layer weight matrix
2: $n' \leftarrow$ desired number of cells
3: $n \leftarrow$ current number of cells
4: while $n > n'$ do
5:     $\vec{F} = \Sigma_j |F_{i,j}|$
6:     $L = argsort(\vec{F})$
7:     for $i = 1...k$ do
8:         $\vec{W_i} = 0$
9:         $n \leftarrow$ current number of cells
10:        if $n == n'$ then
11:            break
12:    re-train the neural network
```

MNIST task in terms of number of parameters, which is the main criteria for model compression. The result indicates that LSTM models are compressible with simple pruning algorithms. Nevertheless, we also find the difference in accuracy performance between dropout and no-dropout models after compression. The difference is potentially due to the information imbalance that is decreased with dropout algorithm during training, so that the neural network is less robust to pruning.

4.3 Contribution and Discussion

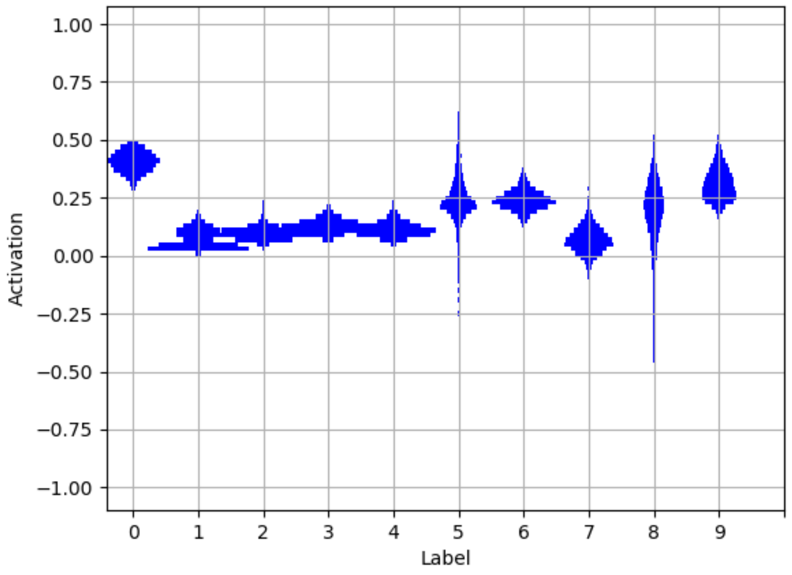
This is one of the first attempts on RNN compression. The proposed algorithms demonstrate the capability of pruning recurrent units. The result could help us understand the memory capacity

# of cells	256	128	64	32	16
# of params	293k	81k	24k	7884	2860
No dropout accuracy	99.06%	98.89%	98.86%	98.55%	95.66%
Dropout accuracy	99.01%	98.97%	98.93%	98.47%	93.94%

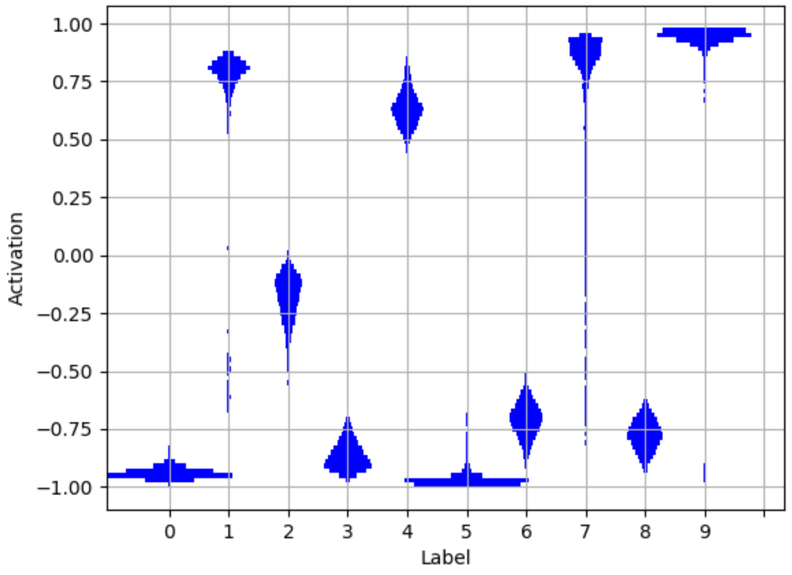
Table 4.1: LSTM compression result

of recurrent units, and dynamic change in hidden states during training. The experiments also imply that the effect of dropout is two-fold. Reduction in redundancy by dropout may weaken the compression capability of the neural network model. Algorithm 2 can be extended with the grow-and-prune paradigm [97] for efficient RNN training.

This work can be extended in many aspects. Firstly, the pruning algorithms can be verified on more tasks and more criteria in the future. Secondly, by finding the extremes of compression, we can further investigate how the information/memory is organized across recurrent layers. Finally, there are many assumptions that need to be mathematically analyzed, and this would be a long-term research topic towards interpretable machine learning.

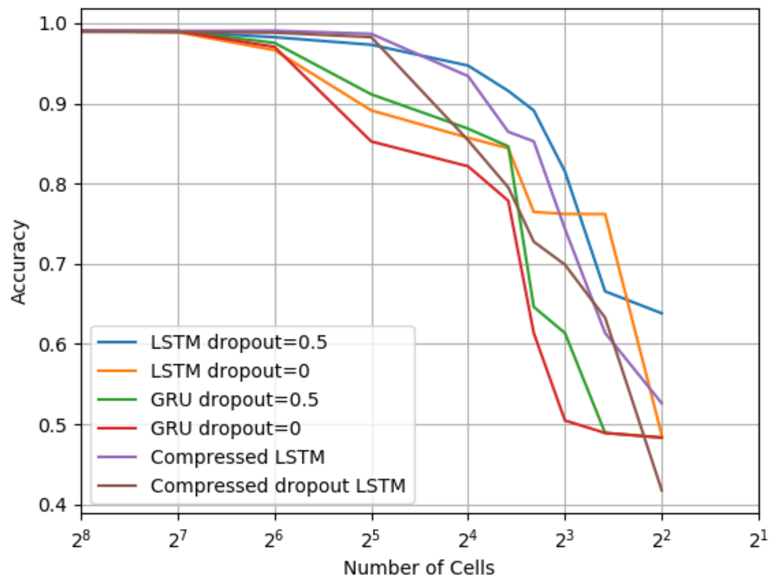


(a) The hidden state histograms of trained LSTM cells with the smallest l_1-norm.

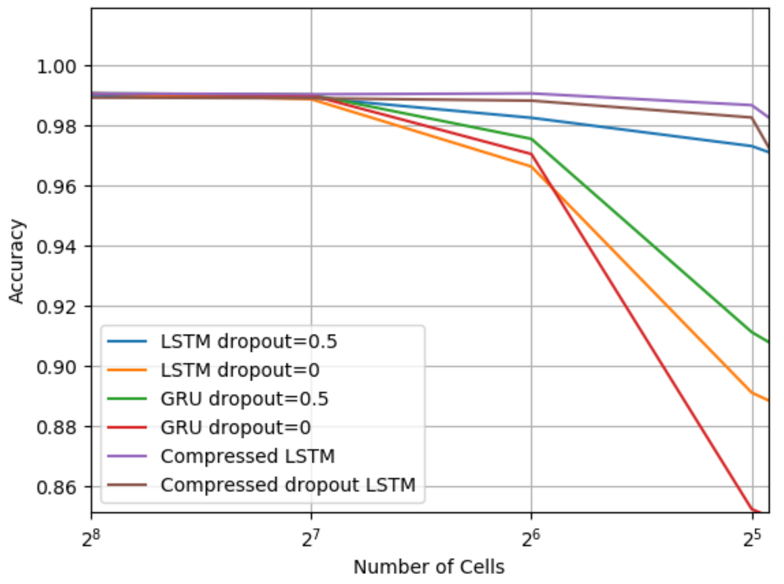


(b) The hidden state histograms of trained LSTM cells with the largest l_1-norm.

Figure 4.1: The hidden state histograms of trained LSTM cells with smallest and largest l_1-norm on all training data of sequential MNIST task.



(a) RNN pruning with algorithm 1, compared with other baseline results



(b) A close look within the range between 256 and 32 cells

Figure 4.2: Accuracy diagram. Models are pruned from the 256-cell LSTM models for sequential MNIST task.

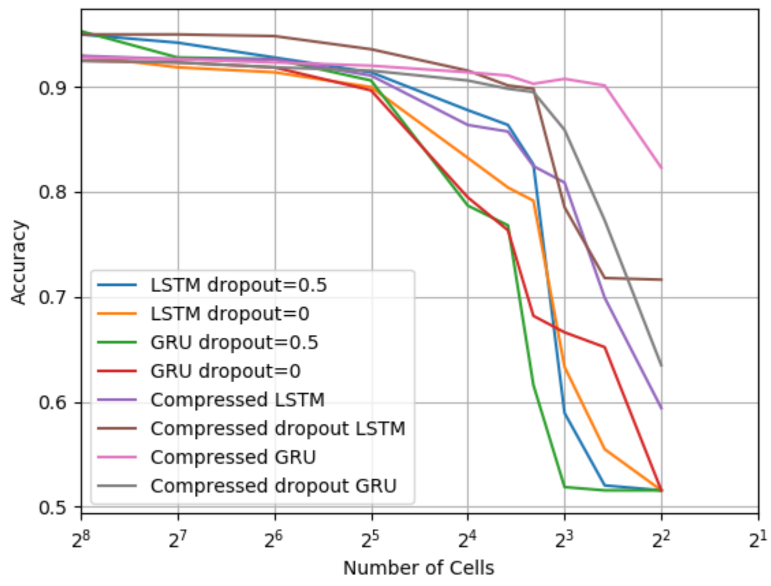


Figure 4.3: Accuracy diagram. Models are pruned from the 256-cell LSTM models for DNA classification task.

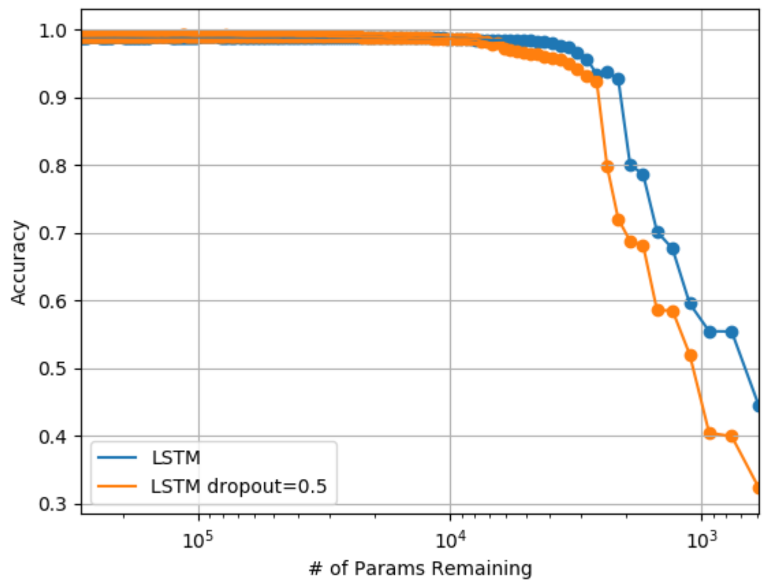


Figure 4.4: Accuracy vs. number of model parameters from LSTM on sequential MNIST task, with x-axis in log scale

5. CONCLUSIONS

With the recent advancement in deep neural network research, the bottlenecks of deep learning also become more significant and urgent. Recurrent neural networks (RNNs) is one kind of powerful machine learning tool for sequence modeling [98], but the lack of interpretability has become one of the main limitations for their broader application. Despite effective methods [99][100] on interpreting the feed-forward neural networks, the complicated hidden dynamics inside recurrent neural networks [101] makes the interpretation of the RNNs extremely difficult.

In this dissertation, I proposed two approaches for understanding the correspondence between the hidden states and the behaviors of recurrent neural networks. The first approach is to represent the recurrent neural network as equivalent dynamical system. By the mathematical analysis on the dynamical system, correspondence can be found between the recurrent model's external behavior and the internal hidden states, such as convergence, periodic orbits, bifurcations, and chaos. This approach can provide deterministic analytical solutions on hidden state stability of RNNs with arbitrary topology. However, the analytical solution from a non-autonomous dynamical system might be difficult to acquire since the dynamical analysis on fiber bundles is computationally expensive.

Another approach is to statistically measure the distributions in both hidden state space and the output space. The learning procedure can be interpreted as the change of distributional representation in the hidden state space. A well-trained network can project the input sequence into different clusters that are significantly separated in the hidden state space. The trajectories in hidden state space also reveal that the recurrent neural networks are robust to partial input as well as lesion to the network topology. The statistical study on LSTM/GRU cells indicate that a single recurrent unit is capable of learning complicated time-dependent patterns. For example, in the sequential MNIST task, the distributional representation of hand-written digits in hidden state space demonstrates similarity with human perception, that similar digits are closer while very different digits can have far distance in the single cell hidden state space.

Based on the above findings, as well as inspired by the convolutional neural network (CNN) compression [20][102], I proposed algorithms for recurrent neural network pruning. The baseline pruning algorithm demonstrates high efficiency and negligible performance loss. This is one of the earliest attempts on RNN compression [87][103]. The compressed RNN models are easier to be deployed on low-resource devices such as mobile phones [104], and smart home controllers [105].

In summary, the dynamical analysis of recurrent neural networks explores the RNN black-box model, and demonstrates how the external behavior is related with the hidden states of the models, and how the learning procedure takes place in the recurrent layer. These findings can help improve the understanding of the recurrent neural network models, and move us one step closer to interpretable machine learning [72].

REFERENCES

[1] C. Olah, "Understanding lstm networks," 2015.

[2] G. Drakos, "What is a recurrent neural networks (rnns) and gated recurrent unit (grus)," 2015.

[3] H. Wang, Q. Li, J. Yoo, and Y. Choe, "Dynamical analysis of recurrent neural circuits in articulated limb controllers for tool use," in *Neural Networks (IJCNN), 2016 International Joint Conference on*, pp. 4339–4345, IEEE, 2016.

[4] S. Kadam, S. Gajbhiye, and R. Banavar, "A geometric approach to the dynamics of flapping wing micro aerial vehicles: Modelling and reduction," *arXiv preprint arXiv:1511.00799*, 2015.

[5] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[6] D. O. Hebb, *The organization of behavior: A neuropsychological theory*. Psychology Press, 2005.

[7] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the national academy of sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

[8] G. E. Hinton, T. J. Sejnowski, *et al.*, "Learning and relearning in boltzmann machines," *Parallel distributed processing: Explorations in the microstructure of cognition*, vol. 1, no. 282-317, p. 2, 1986.

[9] P. J. Werbos, *The roots of backpropagation: from ordered derivatives to neural networks and political forecasting*, vol. 1. John Wiley & Sons, 1994.

[10] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[11] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *nature*, vol. 521, no. 7553, p. 436, 2015.

[12] Y. Bengio *et al.*, “Learning deep architectures for ai,” *Foundations and trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[13] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[14] Z. C. Lipton, “The mythos of model interpretability,” *arXiv preprint arXiv:1606.03490*, 2016.

[15] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.

[16] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, “Squad: 100,000+ questions for machine comprehension of text,” *arXiv preprint arXiv:1606.05250*, 2016.

[17] M.-T. Luong, H. Pham, and C. D. Manning, “Effective approaches to attention-based neural machine translation,” *arXiv preprint arXiv:1508.04025*, 2015.

[18] S. Ruder, “An overview of multi-task learning in deep neural networks,” *arXiv preprint arXiv:1706.05098*, 2017.

[19] S. Han, J. Pool, J. Tran, and W. Dally, “Learning both weights and connections for efficient neural network,” in *Advances in neural information processing systems*, pp. 1135–1143, 2015.

[20] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” *arXiv preprint arXiv:1510.00149*, 2015.

[21] B. Zoph and Q. V. Le, “Neural architecture search with reinforcement learning,” *arXiv preprint arXiv:1611.01578*, 2016.

[22] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," *arXiv preprint arXiv:1802.03268*, 2018.

[23] P. J. Werbos *et al.*, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[24] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[25] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[26] Q. V. Le, N. Jaitly, and G. E. Hinton, "A simple way to initialize recurrent networks of rectified linear units," *arXiv preprint arXiv:1504.00941*, 2015.

[27] M. Arjovsky, A. Shah, and Y. Bengio, "Unitary evolution recurrent neural networks," in *International Conference on Machine Learning*, pp. 1120–1128, 2016.

[28] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[29] F. J. Gomez and R. Miikkulainen, "Solving non-markovian control tasks with neuroevolution," in *IJCAI*, vol. 99, pp. 1356–1361, 1999.

[30] K. O. Stanley and R. Miikkulainen, "Efficient reinforcement learning through evolving neural network topologies," in *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, pp. 569–577, Morgan Kaufmann Publishers Inc., 2002.

[31] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, "Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning," *arXiv preprint arXiv:1712.06567*, 2017.

[32] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy, *et al.*, "Evolving deep neural networks," in *Artificial*

Intelligence in the Age of Neural Networks and Brain Computing, pp. 293–312, Elsevier, 2019.

[33] W. Maass, "Liquid state machines: motivation, theory, and applications," in *Computability in context: computation and logic in the real world*, pp. 275–296, World Scientific, 2011.

[34] H. Jaeger, "Echo state network," *Scholarpedia*, vol. 2, no. 9, p. 2330, 2007.

[35] M. Lukoševičius and H. Jaeger, "Reservoir computing approaches to recurrent neural network training," *Computer Science Review*, vol. 3, no. 3, pp. 127–149, 2009.

[36] N. Bertschinger and T. Natschläger, "Real-time computation at the edge of chaos in recurrent neural networks," *Neural computation*, vol. 16, no. 7, pp. 1413–1436, 2004.

[37] Y. Zhang, P. Li, Y. Jin, and Y. Choe, "A digital liquid state machine with biologically inspired learning and its application to speech recognition," *IEEE transactions on neural networks and learning systems*, vol. 26, no. 11, pp. 2635–2649, 2015.

[38] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 10, pp. 2222–2232, 2016.

[39] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*, pp. 675–678, ACM, 2014.

[40] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pp. 265–283, 2016.

[41] A. Paszke, S. Gross, S. Chintala, and G. Chanan, "Pytorch: Tensors and dynamic neural networks in python with strong gpu acceleration," *PyTorch: Tensors and dynamic neural networks in Python with strong GPU acceleration*, vol. 6, 2017.

[42] J.-C. Latombe, *Robot motion planning*, vol. 124. Springer Science & Business Media, 2012.

[43] A. Graves, A.-r. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE international conference on acoustics, speech and signal processing*, pp. 6645–6649, IEEE, 2013.

[44] R. D. Beer, "A dynamical systems perspective on agent-environment interaction," *Artificial intelligence*, vol. 72, no. 1-2, pp. 173–215, 1995.

[45] R. D. Beer, "On the dynamics of small continuous-time recurrent neural networks," *Adaptive Behavior*, vol. 3, no. 4, pp. 469–509, 1995.

[46] Y. Ilyashenko, "Centennial history of hilbert's 16th problem," *Bulletin of the American Mathematical Society*, vol. 39, no. 3, pp. 301–354, 2002.

[47] A. Whiten, J. Goodall, W. C. McGrew, T. Nishida, V. Reynolds, Y. Sugiyama, C. E. Tutin, R. W. Wrangham, and C. Boesch, "Cultures in chimpanzees," *Nature*, vol. 399, no. 6737, p. 682, 1999.

[48] C. Boesch and H. Boesch, "Tool use and tool making in wild chimpanzees," *Folia primatologica*, vol. 54, no. 1-2, pp. 86–99, 1990.

[49] P. Foerder, M. Galloway, T. Barthel, D. E. Moore III, and D. Reiss, "Insightful problem solving in an asian elephant," *PloS one*, vol. 6, no. 8, p. e23251, 2011.

[50] J. Boswall, "Tool-using and related behaviour in birds: More notes," *Aviculture Magazine*, vol. 89, pp. 94–108, 1983.

[51] J. R. Chung and Y. Choe, "Emergence of memory in reactive agents equipped with environmental markers," *IEEE Transactions on Autonomous Mental Development*, vol. 3, no. 3, pp. 257–271, 2011.

[52] Q. Li, J. Yoo, and Y. Choe, "Emergence of tool use in an articulated limb controlled by evolved neural circuits," in *Neural Networks (IJCNN), 2015 International Joint Conference on*, pp. 1–8, IEEE, 2015.

[53] T. A. Mann and Y. Choe, "Prenatal to postnatal transfer of motor skills through motor-compatible sensory representations," in *2010 IEEE 9th International Conference on Development and Learning*, pp. 185–190, IEEE, 2010.

[54] R. Murphy and R. R. Murphy, *Introduction to AI robotics*. MIT press, 2000.

[55] C. H. Papadimitriou and N. K. Vishnoi, "On the computational complexity of limit cycles in dynamical systems," *arXiv preprint arXiv:1511.07605*, 2015.

[56] A. M. Lyapunov, "The general problem of the stability of motion," *International journal of control*, vol. 55, no. 3, pp. 531–534, 1992.

[57] C. Yu and D. H. Ballard, "On the integration of grounding language and learning objects," in *AAAI*, vol. 4, pp. 488–493, 2004.

[58] J. Modayil and B. Kuipers, "Autonomous development of a grounded object ontology by a learning robot," in *Proceedings of the national conference on Artificial intelligence*, vol. 22, p. 1095, Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007.

[59] L. Johnson and D. H. Ballard, "Efficient codes for inverse dynamics during walking," in *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

[60] J. Lehman and K. O. Stanley, "Abandoning objectives: Evolution through the search for novelty alone," *Evolutionary computation*, vol. 19, no. 2, pp. 189–223, 2011.

[61] R. D. Beer, H. J. Chiel, R. D. Quinn, K. S. Espenschied, and P. Larsson, "A distributed neural network architecture for hexapod robot locomotion," *Neural Computation*, vol. 4, no. 3, pp. 356–365, 1992.

[62] J. D. Hamilton, *Time series analysis*, vol. 2. Princeton university press Princeton, NJ, 1994.

[63] P. Coulibaly and N. Evora, "Comparison of neural network methods for infilling missing daily weather records," *Journal of hydrology*, vol. 341, no. 1-2, pp. 27–41, 2007.

[64] I. Maqsood, M. R. Khan, G. H. Huang, and R. Abdalla, "Application of soft computing models to hourly weather analysis in southern saskatchewan, canada," *Engineering Applications of Artificial Intelligence*, vol. 18, no. 1, pp. 115–125, 2005.

[65] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, *et al.*, "Deep speech: Scaling up end-to-end speech recognition," *arXiv preprint arXiv:1412.5567*, 2014.

[66] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber, "A novel connectionist system for unconstrained handwriting recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 31, no. 5, pp. 855–868, 2008.

[67] A. Graves, M. Liwicki, H. Bunke, J. Schmidhuber, and S. Fernández, "Unconstrained online handwriting recognition with recurrent neural networks," in *Advances in neural information processing systems*, pp. 577–584, 2008.

[68] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[69] J. C. Becker and D. H. Gottlieb, "The transfer map and fiber bundles," *Topology*, vol. 14, no. 1, pp. 1–12, 1975.

[70] S. Pradhan, A. Hansen, and B. K. Chakrabarti, "Failure processes in elastic fiber bundles," *Reviews of modern physics*, vol. 82, no. 1, p. 499, 2010.

[71] S. Kobayashi and K. Nomizu, *Foundations of differential geometry*, vol. 1. New York, 1963.

[72] C. Molnar, *Interpretable Machine Learning*. 2019. `https://christophm.github.io/interpretable-ml-book/`.

[73] V. Bewick, L. Cheek, and J. Ball, "Statistics review 7: Correlation and regression," *Critical care*, vol. 7, no. 6, p. 451, 2003.

[74] C. Chakraborty and P. Talukdar, “Issues and limitations of hmm in speech processing: a survey,” *International Journal of Computer Applications*, vol. 141, no. 7, pp. 975–8887, 2016.

[75] D. Dua and C. Graff, “UCI machine learning repository,” 2017.

[76] C. L. Blake and C. J. Merz, “Uci repository of machine learning databases, 1998,” 1998.

[77] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[78] L. Bottou, “Large-scale machine learning with stochastic gradient descent,” in *Proceedings of COMPSTAT’2010*, pp. 177–186, Springer, 2010.

[79] J. Platt *et al.*, “Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods,” *Advances in large margin classifiers*, vol. 10, no. 3, pp. 61–74, 1999.

[80] S. Kullback and R. A. Leibler, “On information and sufficiency,” *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[81] E. D. Sontag and Y. Wang, “On characterizations of the input-to-state stability property,” *Systems & Control Letters*, vol. 24, no. 5, pp. 351–359, 1995.

[82] L. v. d. Maaten and G. Hinton, “Visualizing data using t-sne,” *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[83] H. Hotelling, “Analysis of a complex of statistical variables into principal components.,” *Journal of educational psychology*, vol. 24, no. 6, p. 417, 1933.

[84] C.-Y. Liou, J.-C. Huang, and W.-C. Yang, “Modeling word perception using the elman network,” *Neurocomputing*, vol. 71, no. 16-18, pp. 3150–3157, 2008.

[85] Y. C. Eldar and G. Kutyniok, *Compressed sensing: theory and applications*. Cambridge university press, 2012.

[86] E. B. Baum and D. Haussler, "What size net gives valid generalization?," in *Advances in neural information processing systems*, pp. 81–90, 1989.

[87] S. Narang, E. Elsen, G. Diamos, and S. Sengupta, "Exploring sparsity in recurrent neural networks," *arXiv preprint arXiv:1704.05119*, 2017.

[88] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in neural information processing systems*, pp. 598–605, 1990.

[89] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016.

[90] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient transfer learning," *arXiv preprint arXiv:1611.06440*, vol. 3, 2016.

[91] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," *arXiv preprint arXiv:1803.03635*, 2018.

[92] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless cnns with low-precision weights," *arXiv preprint arXiv:1702.03044*, 2017.

[93] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[94] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in *International Conference on Machine Learning*, pp. 2285–2294, 2015.

[95] L. Rüschendorf, "The wasserstein distance and approximation theorems," *Probability Theory and Related Fields*, vol. 70, no. 1, pp. 117–129, 1985.

[96] D. G. Simpson, "Minimum hellinger distance estimation for the analysis of count data," *Journal of the American statistical Association*, vol. 82, no. 399, pp. 802–807, 1987.

[97] X. Dai, H. Yin, and N. K. Jha, "Incremental learning using a grow-and-prune paradigm with efficient neural networks," *arXiv preprint arXiv:1905.10952*, 2019.

[98] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[99] Q. Zhang, Y. Yang, H. Ma, and Y. N. Wu, "Interpreting cnns via decision trees," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6261–6270, 2019.

[100] B. Zhou, D. Bau, A. Oliva, and A. Torralba, "Interpreting deep visual representations via network dissection," *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[101] H. Strobelt, S. Gehrmann, B. Huber, H. Pfister, A. M. Rush, *et al.*, "Visual analysis of hidden state dynamics in recurrent neural networks," *arXiv preprint arXiv:1606.07461*, 2016.

[102] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[103] M. Zhu and S. Gupta, "To prune, or not to prune: exploring the efficacy of pruning for model compression," *arXiv preprint arXiv:1710.01878*, 2017.

[104] G. Chen, C. Parada, and G. Heigold, "Small-footprint keyword spotting using deep neural networks," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4087–4091, IEEE, 2014.

[105] B. Li, T. N. Sainath, A. Narayanan, J. Caroselli, M. Bacchiani, A. Misra, I. Shafran, H. Sak, G. Pundak, K. K. Chin, *et al.*, "Acoustic modeling for google home.," in *Interspeech*, pp. 399–403, 2017.

APPENDIX A

SUPPLEMENTAL PLOTS

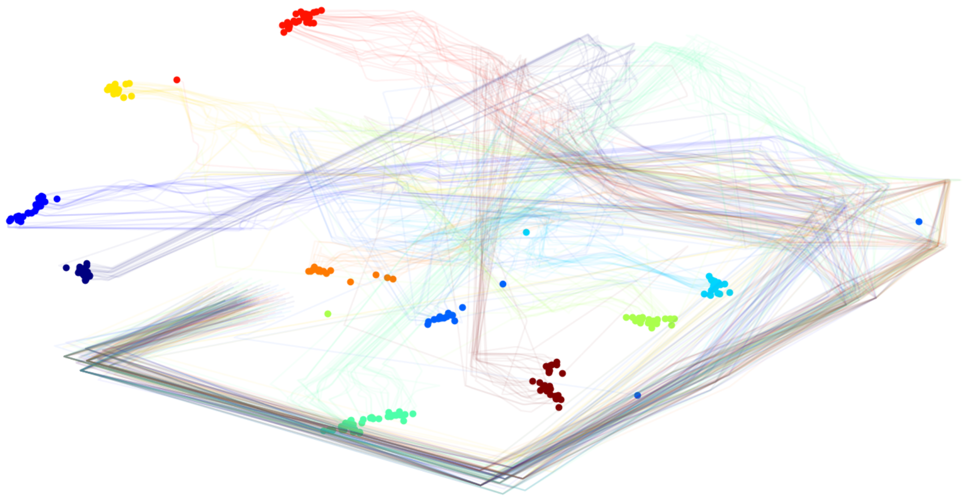


Figure A.1: 2-D t-SNE of the hidden states in a 256-cell LSTM classifier

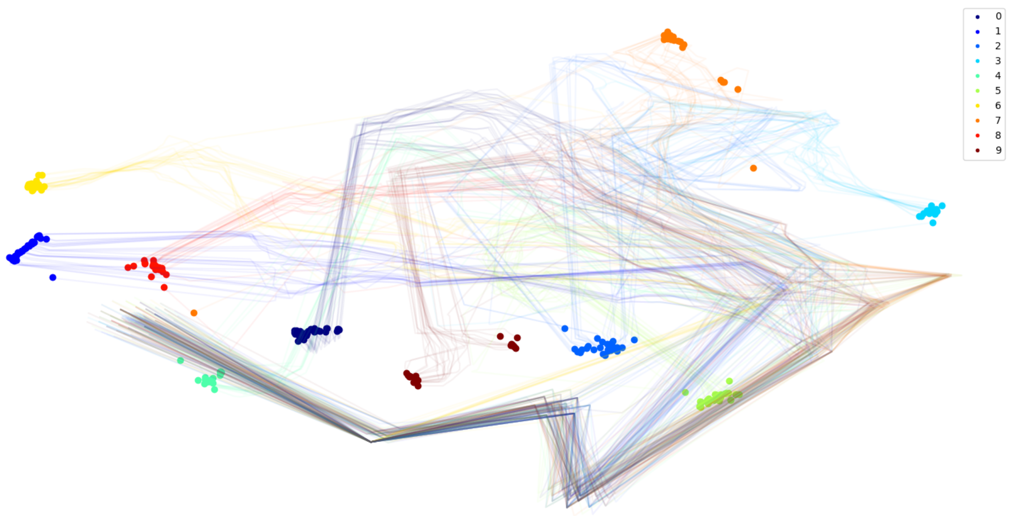


Figure A.2: 2-D t-SNE of the hidden states in a 256-cell LSTM classifier, using another subset of MNIST input.

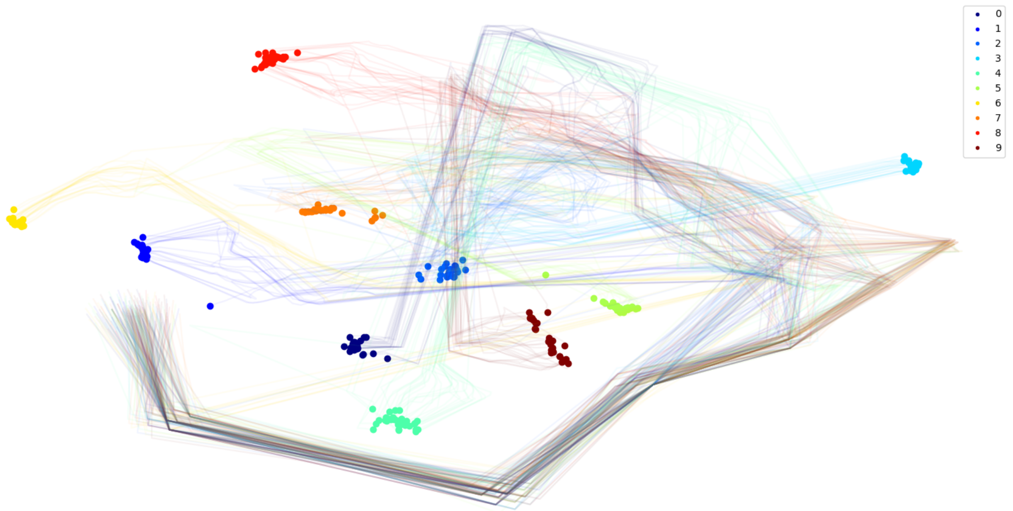


Figure A.3: 2-D t-SNE of the hidden states in a 256-cell LSTM classifier, using another subset of input.

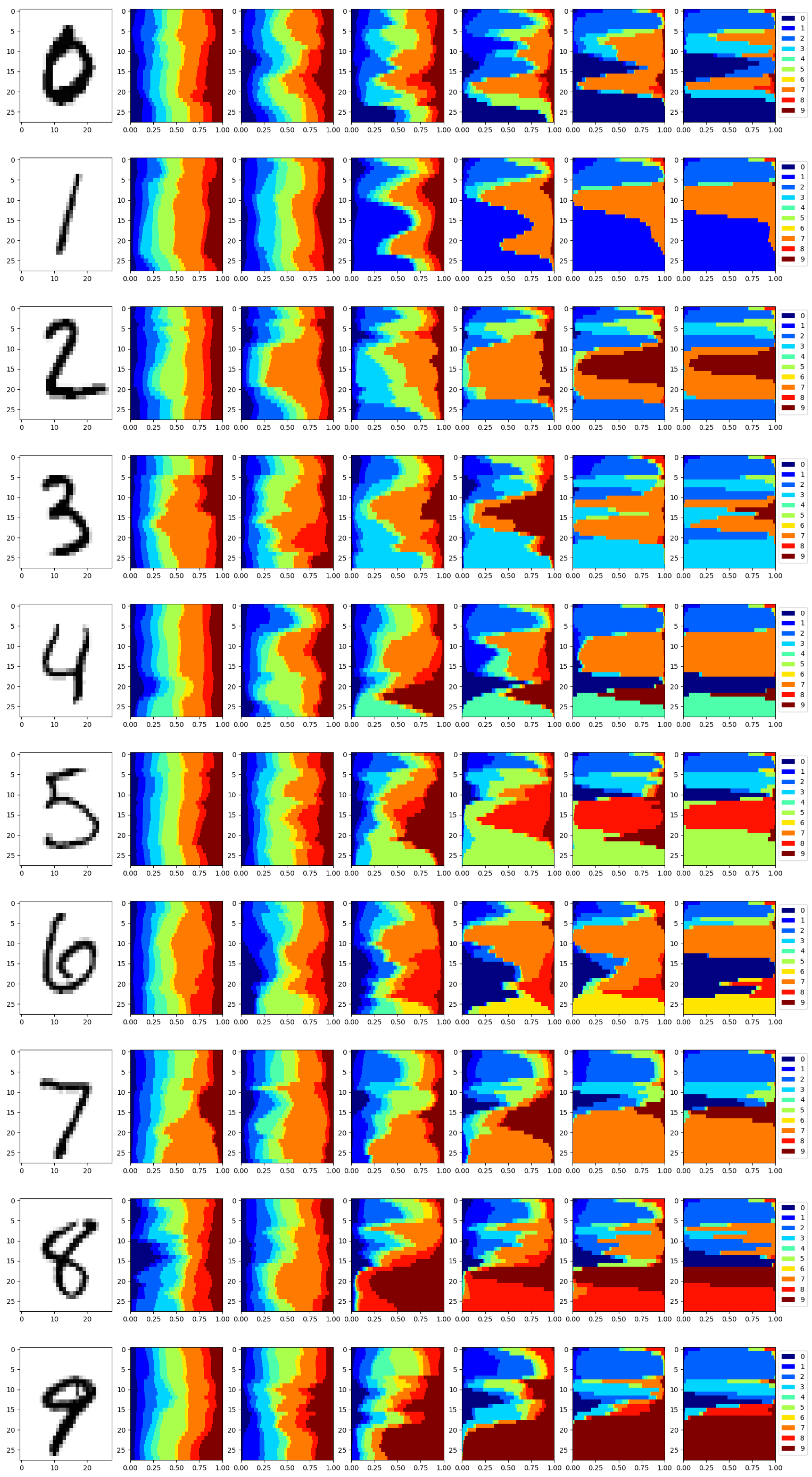


Figure A.4: Examples of the input images and the corresponding sequential representations. Starting from step 0 to step 27, the colored stacked histograms demonstrate the changes in softmax prediction vectors. From left to right, each column corresponds to the trained model with 8, 16, 32, 64, 128, 256 LSTM cells.