12-2014

Scalable Hardware Efficient Deep Spatio-Temporal Inference Networks

Steven Robert Young
University of Tennessee - Knoxville, syoung22@vols.utk.edu

To the Graduate Council:

I am submitting herewith a dissertation written by Steven Robert Young entitled "Scalable Hardware Efficient Deep Spatio-Temporal Inference Networks." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Computer Engineering.

Itamar Arel, Major Professor

We have read this dissertation and recommend its acceptance:

Jens Gregor, Jeremy Holleman, Xiaopeng Zhao

Accepted for the Council:

Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

12-2014

Scalable Hardware Efficient Deep Spatio-Temporal Inference Networks

Steven Robert Young
University of Tennessee - Knoxville, syoung22@vols.utk.edu

To the Graduate Council:

I am submitting herewith a dissertation written by Steven Robert Young entitled "Scalable Hardware Efficient Deep Spatio-Temporal Inference Networks." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Computer Engineering.

Itamar Arel, Major Professor

We have read this dissertation and recommend its acceptance:

Jens Gregor, Jeremy Holleman, Xiaopeng Zhao

Accepted for the Council:
Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

Scalable Hardware Efficient Deep Spatio-Temporal Inference Networks

A Dissertation Presented for the

Doctor of Philosophy

Degree

The University of Tennessee, Knoxville

Steven Robert Young

December 2014

Dedicated to my wife Angela. Thank you for your endless support and encouragement during my academic pursuits.

Acknowledgements

I would like to thank Derek Rose, Andrew Davis, Ben Goodrich, Benjamin Martin, Aaron Mishtal, Tom Karnowski, and Bobby Coop for providing thought provoking research discussion in the lab during my studies and Junjie Lu for his great help and support in helping me understand the effects of analog circuitry upon computation. I would also like to thank my advisor Dr. Itamar Arel for providing me the opportunity to pursue my PhD and for his guidance and instruction through my research for the past six years. My committee dissertation committee members, Dr. Jens Gregor, Dr. Jeremy Holleman, and Dr. Xiaopeng Zhao, deserve much gratitude for their time spent reviewing and contributing suggestions to this dissertation. I would like to thank Dr. Michael Roberts for recommending me for an undergraduate research position to Dr. Arel. Without that experience, I would have never seen graduate school as a possibility.

Finally, I must thank my family. My parents, Jay and Teresa, have provided me with the support, love, and opportunities without which this would not be possible. My father-in-law and mother-in-law, Kenneth and Teresa Greene, have welcomed me into their family and provided me with many enjoyable distractions from my work over the past five years. Most importantly, my wife Angela has supported me through the entirety of my graduate studies, even when my work has limited the time we have to enjoy together.

Abstract

Deep machine learning (DML) is a promising field of research that has enjoyed much success in recent years. Two of the predominant deep learning architectures studied in the literature are Convolutional Neural Networks (CNNs) and Deep Belief Networks (DBNs). Both have been successfully applied to many standard benchmarks with a primary focus on machine vision and speech processing domains.

Many real-world applications involve time-varying signals and, consequently, necessitate models that efficiently represent both temporal and spatial attributes. However, neither DBNs nor CNNs are designed to naturally capture temporal dependencies in observed data, often resulting in the inadequate transformation of spatio-temporal signals into wide spatial structures. It is argued that deep machine learning without proper temporal representation mechanisms is unable to extract meaningful information from many time-varying natural signals.

Another clear emerging need is in growing deep learning architectures with the size of the problem at hand, suggesting that such architectures should map well to custom hardware platforms. The latter offer much better performance than that achievable using CPUs or even GPUs. Analog computation is a unique potential solution to the scalability challenge offering the benefits of low power consumption and smaller physical size when compared to digital implementations. However, these benefits come with the consequence of inaccurate computations and noise.

This work presents an enhanced formulation of DeSTIN - a Deep Spatio-Temporal Inference Network (DeSTIN) that is inherently designed to capture both spatial and temporal dependencies in the data provided. The regular structure of DeSTIN, its computational requirements, and local connectivity render it hardware-efficient and highly scalable. Implementation of DeSTIN using analog computation is studied in detail, where the architectural robustness to various distortions in its signals is demonstrated. To the best of our knowledge, this is the first time custom analog hardware has been developed for deep machine learning. Key enhancements to previous formulations of DeSTIN are discussed in detail and results on standard benchmarks are presented. This work helps pave the way for advancing deep learning to address some of the long-standing challenges in machine learning.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this chapter, the field of deep learning will be introduced and the research to be presented throughout the remainder of this dissertation will be outlined. The work presented in this dissertation focuses on a particular deep learning architecture known as Deep Spatio-Temporal Inference Networks (DeSTIN).

1.1 Deep Learning

Deep learning is a field that is currently receiving a large amount of attention. A recent article listed it as a "Top Ten Breakthrough Technology"of 2013 (Hof, 2013), and Google's success in using it to categorize millions of YouTube videos (Le et al., 2012) has put the technology in the spotlight of many news sources. While these success stories have received much attention, deep learning remains a very active area of research. Some of the key challenges still faced by the research community include devising methods to train such architectures, establishing their scalability properties as well as the ability to mapped well to massive parallel architectures. Moreover, inherently capturing temporal dependencies in natural signals remains largely unaddressed.

Developing models that allow computers to achieve *intelligence* has long been a focus of research. In order to work with problems that demonstrate *intelligence*, the systems designed must be able to work with the large amounts of features and high-dimensional data that are associated with these types of problems. In short, these systems must deal with the "curse of dimensionality"(Bellman, 1957), or the fact that the complexity of learning problems grows exponentially with the dimensionality of the data involved. The conventional approach to addressing this challenge involves utilizing human-domain knowledge to map problems to smaller feature spaces - a process which is often difficult and has little application outside very specific domains (Duda et al., 2000).

The definition of a deep architecture is a model that is comprised of multiple layers of non-linear transformations (Bengio, 2009; Bengio et al., 2013; Schmidhuber, 2014). While this definition describes the necessary components of a deep learning method, it does not describe the purpose, use cases, or motivation for deep learning. Deep learning attempts to remove or reduce the need for domain knowledge in machine learning tasks and learn features directly from the data. Deep learning methods are currently the state of the art for many object recognition tasks. They achieve this without any prior information of the task other than the fact that the task is to classify images. They do not pre-configured with the knowledge that certain objects are frequently a particular color or that certain objects contain hard edges. They are not even pre-configured to detect particular patterns in the images such as edges or particular shapes.

Deep learning methods seek to learn a hierarchy of features where features at the higher layers are constructed from features formed at the lower layers (Bengio, 2009). Automatically learning these features with multiple levels of abstraction is the key to avoiding the need for human-engineering of features. The *depth* of the architecture refers to these layers of increasing abstraction, which are based on recent research of the mammalian brain suggesting that the brain learns through the construction of multiple layers of abstraction. This hierarchy of abstraction is the mechanism that allows deep architectures to learn complex representations with little human intervention or knowledge of the problem domain (Bengio and LeCun, 2007).

It is important to note that although deep architectures may appear to overcome the curse of dimensionality that large data presents, in reality they do so by taking advantage of the fact that some structure or locality exists in natural signals, which allows sparsity to exist in the architectures. Without taking advantage of this structure, the problem would still exist. In order to take advantage of the locality, deep learning methods represent relationships between spatially close signals with greater detail while letting distant signals be represented with less detail. This is a direct consequence of the increasing level of abstraction, or decreasing level of detail, that results as signals move up through the hierarchies used. Though much of the current research focuses on representing images, deep learning methods can be used to discover structure of any modality Ngiam et al. (2011) with one successful example being music signals Hamel and Eck (2010) and another is speech recognition (Hinton et al., 2012).

1.2 DeSTIN - A Compositional DML Architecture

This work focuses on DeSTIN - a compositional deep machine learning architecture. Compositional deep learning architectures are a particular family of DML systems characterized by hosting multiple instantiations of a basic cortical circuit (or node) that populates all layers of the hierarchy. Every node learns to represent the sequences of patterns presented to it by a unique subset of nodes in the layer below it. The nodes at the very lowest layer of the hierarchy operate on input data and construct a belief state that tries to characterize the sequences of patterns observed in a compact manner. All the layers above the input layer operate on the belief states of nodes in the layer below and construct their own belief states that capture regularities in this space. The learning process in this type of DML architecture aims to form a hierarchical feature space that can be employed by classification or regression models. The learning process at each node consists of exposing it to a large set of observations and allowing the salient attributes of these observations to be captured. The resulting feature space should exhibit invariance to common distortions and variations in the observations in order for the representations to be robust.

1.3 Benefits of Analog Architectures

The future of deep learning methods will lead to working with datasets of increasing dimensionality. In order to work with these large datasets, the computational complexity and storage requirements involved in the training of DML architectures must be addressed (Erhan et al., 2009). While there has been some work in implementing CNNs using FPGAs (Farabet et al., 2011, 2009, 2010), current research has stopped short of actually creating custom hardware implementations in either analog or digital technology. This research aims to enable deep learning to utilize the power and speed advantages of analog hardware. Custom analog circuitry provides a way to address the limitations of general purpose computers and digital VLSI technology. The physical density of the building blocks of a DML system are critically important to achieving the largest possible system provided a cost or physical size constraint. Analog computation circuitry can provide an implementation that uses one to two orders of magnitude less area than comparable digital technology and often provides a significant reduction in power consumption. Both of these attributes will become ever more important to DML architectures, which are designed to work on large datasets, as they are moved closer to the sensors that provide the data they work on in order to avoid sending large amounts of data over bandwidth limited networks.

In analog computation the natural physics of the devices provides the mechanism for achieving this large improvement in density. For example, when using electric currents to represent values, simple nodal analysis tells us we can perform addition by just wiring these values together. The benefits that can be achieved in area and power do come with disadvantages, such as offset errors, gain errors, and various noise. However, the feedback that exists in most learning systems often helps to naturally compensate for these inaccuracies. The brain is known to be comprised of highly noisy, inaccurate neuron activations and interconnection signals. Many of the behavioral responses of animals occur in time periods around 30 ms long (Bialek et al., 1991), and if neural signals are integrated over comparable time windows, they usually have a signal-to-noise ratio (SNR) in the range of 1-10 (Bialek et al., 1991; de Ruyter van Steveninck et al., 1997; van Rossum et al., 2003). SNRs of this level can easily be achieved in moderate precision analog electronics. The low SNR requirements of neural systems provide an opportunity to relax the accuracy requirements for electronic computational primitives in order to allow aggressive optimization for area and power consumption.

1.4 Contributions

The contributions of this research work include:

- Introduction of a recurrent clustering algorithm that is able to encode temporal information into the belief states of DeSTIN nodes.
- A supervised training scheme for DeSTIN that drives it to not only discover regularities, but to focus on regularities relevant to the task at hand.
- A modified DeSTIN architecture - Convolutional DeSTIN - that allows DeSTIN to be applied to natural images instead of being confined to more structured handwritten digit images.
- A DeSTIN architecture that maps well to parallel and custom analog circuitry both in theory, such that nodes can operate in parallel and leverage the benefits of analog design, and in practice, such that an implementation using these technologies is a practical task.
- A detailed study of the effect inaccuracies introduced by analog computational elements have upon custom DeSTIN architecture realizations.

1.5 Publications

The following publications have appeared, addressing various parts of the research conducted in this work.

- S. Young, J. Lu, J. Holleman, I. Arel "On the Impact of Approximate Computation in Analog Deep Machine Learning," IEEE Transactions on Neural Networks and Learning Systems, May 2014
- S. Young, A. Davis, A. Mishtal, I. Arel, "Hierarchical Spatiotemporal Feature Extraction using Recurrent Online Clustering," Pattern Recognition Letters, January, 2014
- S. Young, I. Arel, "Recurrent Clustering for Unsupervised Feature Extraction with Application to Sequence Detection," in Proc. IEEE International Conference on Machine Learning and Applications (ICMLA), December, 2012
- S. Young, I. Arel, T. Karnowski, D. Rose, "A Fast and Stable Incremental Clustering Algorithm," in Proc. 7th International Conference on Information Technology (ITNG), April, 2010

Additionally, this work has supported the design of an analog clustering chip and a small-scale analog DeSTIN implementation presented in the following publications.

- J. Lu, S. Young, I. Arel, J. Holleman, "A 1 TOPS/W Analog Deep Machine-Learning Engine with Floating-Gate Storage in 0.13 um CMOS," to appear in IEEE Journal of Solid State Circuits, January, 2015
- J. Lu, S. Young, I. Arel, J. Holleman, "A 1TOPS/W Analog Deep Machine Learning Engine with Floating-Gate Storage in 0.13um CMOS," in Proc. 2014 IEEE international Solid-State Circuits Conference (ISSCC), February, 2014

- J. Lu, S. Young, I. Arel, J. Holleman, "An Analog Online Clustering Circuit in 130nm CMOS," in Proc. IEEE Asian Solid-State Circuits Conference, November, 2013

1.6 Dissertation Outline

In Chapter 2, an introduction to machine learning techniques necessary to understand deep machine learning and DeSTIN is provided along with an overview of the primary deep learning architectures studied in the literature. Chapter 3 introduces DeSTIN and the contributions made in enhancing it relative to its previous formulation. Chapter 4 discusses the implications of implementing DeSTIN in custom analog hardware. Experimental results are presented in Chapter 5, which highlight the attributes of DeSTIN and demonstrate the performance capabilities of DeSTIN on standard benchmarks. Finally, in Chapter 6 a discussion of the impact of the work is given along with some proposals for future work.

Chapter 2

Background and Literature Review

2.1 Machine Learning

Machine learning (ML) can be defined as the field of study surrounding machines that can *learn* to perform a task. *Learning* is the mechanism that distinguishes the field from others. If a machine can perform a task before it is provided with data just as well as after it is provided with data, it is not learning. This suggests that ML algorithms must improve their performance with experience (Mitchell, 1997), rather than its designers simply creating a system that is limited to their experiences.

Programming computers to learn is an appealing idea, since computers have the ability to store large amounts of data very precisely, the ability to perform calculations very quickly, and cost very little to acquire or use. These abilities have successfully been leveraged to allow computers to perform tasks better than any human can, such as the chess playing computer Deep Blue (Campbell et al., 2002). ML systems have also been able to complete tasks very well that no human could reasonably complete, such as the web search engine Google (Brin and Page, 1998) that spawned the company of the same name. However, these systems were only designed to perform very specific tasks, and many of the mechanisms that make them work do not carry over into general machine learning tasks. The focus of the rest of the work presented here will be on more general ML algorithms that are designed to perform a variety of tasks.

2.1.1 Unsupervised Learning

Unsupervised learning consists of methods in which there is no explicit teacher (Duda et al., 2000). This type of algorithm strives to find structure in the dataset it is provided without the help of any labeled data. While attempting to learn from data without having context provided via labeled data may seem like an undesirable goal, there are many benefits to unsupervised learning. The most obvious one being that labeling large datasets is often prohibitively expensive. Unsupervised learning algorithms are also able to take a complex, high-dimensional space and map it to a simpler, lower-dimensional space. Thus, unsupervised learning methods are often used to make classification a simpler task or to improve classification results altogether, and not directly for classification. The remainder of this section outlines some important unsupervised methods.

Clustering

The type of clustering that will be discussed here is unsupervised clustering which strives to partition the feature space in order to characterize the data without any labeled data guiding the process. Clustering operates on the premise that similar data-points share meaning. Thus, it attempts to split the dataset into groups of data-points that are spatially close to one another in order to assign meaning to a data-point based on which partition it lies in.

The most widely used method of clustering is k-means clustering (MacQueen, 1967). This method attempts to select a set of centroids (or set means), $\boldsymbol{\mu}$, that partition the space the observations, $\mathbf{o}$, into k sets, $\mathbf{S}$, in such a manner as to minimize the within-cluster sum of squares:

$$\underset{\mathbf{S}}{\operatorname{argmin}} \sum_{i=1}^{k} \sum_{o_j \in S_i} \parallel x_j - \mu_i \parallel^2 \tag{2.1}$$

The standard algorithm for performing k-means clustering is straightforward:

1. Start with some initial set of centroids $\boldsymbol{\mu}$.

2. Assign each point to the set S_i with the nearest centroid μ_i. (Assignment Rule)

3. Calculate the means (new centroids) $\boldsymbol{\mu}$ of the resulting sets. (Update Rule)

4. Repeat steps 2 and 3 until the assignments do not change.

This process requires that the distance between each centroid and the points in its set be calculated each time the centroids are updated which causes the method to be computationally intensive. If the size of the dataset makes these calculations impractical or the complete dataset is not available such as in a real-time system, an incremental (or online) update method can be used. Adapting k-means to have an incremental update rule results in the following algorithm:

1. Start with some initial set of centroids $\boldsymbol{\mu}$.

2. Find the nearest centroid, $\boldsymbol{\mu}_j$ to the current observation, o_j.

$$i \leftarrow \underset{i}{argmin} \parallel o_j - \mu_i \parallel \tag{2.2}$$

3. Update the winning centroid, μ_i, towards the current observation, o_j, by some learning rate, η.

$$\mu_i \leftarrow \mu_i + \eta \left(o_j - \mu_i \right) \tag{2.3}$$

4. Repeat steps 2 and 3 until some stopping criteria is met.

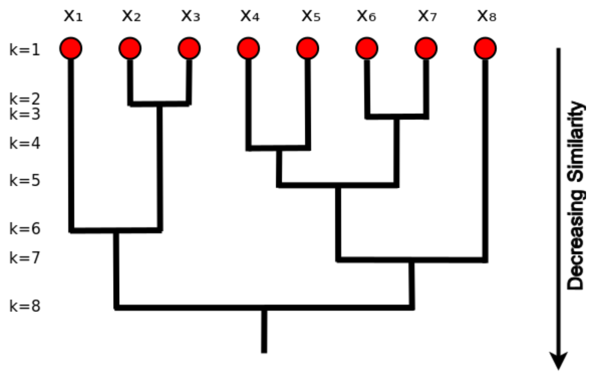


Figure 2.1: This dendrogram represents the results of hierarchical clustering.

Algorithm 1 Agglomerative Hierarchical Clustering

```
1: $c, \hat{c} \leftarrow n, \mathbb{D} \leftarrow x_i, i = 1, \dots, n$
2: while $c < \hat{c}$ do
3:     $\hat{c} \leftarrow \hat{c} - 1$
4:     find nearest clusters, $\mathbb{D}_j$ and $\mathbb{D}_k$
5:     merge $\mathbb{D}_j$ and $\mathbb{D}_k$
6: end while
```

This method is known as Winner-Take-All (WTA) clustering, or competitive learning, since only a single centroid, the one closest to the current observation, is updated for each data point (Duda et al., 2000). The learning rate, η, is particularly important. If η is held constant over time, it can cause the centroids to never settle to a fixed value. If η decays over time in order to force convergence, the centroids may not be able to represent novel patterns observed later in the sequence or it may not be able to track gradual, continuing changes in the data.

Another type of clustering often seen is hierarchical clustering (Day and Edelsbrunner, 1983). This method of clustering is based on the idea that clusters have sub-clusters (Duda et al., 2000). In agglomerative hierarchical clustering, each data point, x_i starts out as its own cluster $\mathbb{D}_i$. The nearest two clusters are merged to form their own cluster and then the process is repeated until the model has been reduced to the specified number, c, of clusters is produced. This process, outlined in Algorithm 1, can be represented with a dendrogram as shown in Figure 2.1.

Gaussian Mixture Models

A Gaussian Mixture Model uses a weighted sum of Gaussian components to represent a probability density function (Reynolds, 2008). A GMM made up of M components is given by

$$p\left(x|\lambda\right) = \sum_{i=1}^{M} w_i g\left(x|\mu_i, \Sigma_i\right) \tag{2.4}$$

where x is a D-dimensional vector, w_i is the mixture weight for component i, and $g\left(x|\mu_i, \Sigma_i\right)$ is the component density which is modeled as a D-variate Gaussian distribution, $\sum_{i=1}^{M} w_i = 1$, and $\lambda = \{\mathbf{w}, \boldsymbol{\mu}, \boldsymbol{\Sigma}\}$. With a sufficient number of components, GMMs can be used to model any continuous distribution.

The goal of training a GMM is to find the most likely model given the set of training vectors. In order to train the GMM using T training vectors X, the GMM likelihood

$$p\left(X|\lambda\right) = \prod_{t=1}^{T} p\left(x_t|\lambda\right) \tag{2.5}$$

needs to be maximized, assuming the training vectors are independent. This independence assumption means only the diagonal elements of Σ_i will be non-zero. The optimum parameters can be obtained iteratively using the following special case of the expectation-maximization (EM) algorithm.

$$w_i = \frac{1}{T} \sum_{t=1}^{T} Pr(i|x_t, \lambda) \tag{2.6}$$

$$\mu_i = \frac{\sum_{t=1}^{T} \left[Pr(i|x_t, \lambda) x_t\right]}{\sum_{t=1}^{T} Pr(i|x_t, \lambda)} \tag{2.7}$$

$$\sigma_i^2 = \frac{\sum_{t=1}^{T} \left[Pr(i|x_t, \lambda) x_t^2\right]}{\sum_{t=1}^{T} Pr(i|x_t, \lambda)} - \mu_i^2 \tag{2.8}$$

$$Pr(i|x_t, \lambda) = \frac{w_i g(x_t|\mu_i, \Sigma_i)}{\sum_{k=1}^{M} w_k g(x_t|\mu_k, \Sigma_k)} \tag{2.9}$$

2.1.2 Supervised Learning

Supervised learning methods use labeled data in order to train a classifier that can predict the label of unlabeled test data. More formally, they attempt to model the posteriori probability $P(\omega_i|x)$ for class ω_i and the data point x. Performance of these classifiers is typically measured as the percentage of correctly labeled data samples from the test set. Supervised learning methods can be separated into two main types: parametric techniques and non-parametric techniques. Parametric techniques make some assumption about the nature of the data (e.g. it obeys Gaussian distributions) and learns a set of parameters to fit the data. Non-parametric techniques make no assumption about the nature of the data. Since non-parametric techniques are the type often paired with DeSTIN and other DML methods, they will be the focus of the remainder of this section.

k-Nearest Neighbor Classifier

The k-NN classifier operates by assigning the test point x the label most frequently represented among the k nearest samples (Duda et al., 2000). This rule for determining the most likely label for test data can be derived rather simply. If k is the fixed value referring to the number of points in a sphere of volume V and n is the total number of points in the dataset, then the probability density for x is $p(x) = \frac{k}{nV}$. This means that the joint probability for the data point x and each class ω_i is given by

$$p(x, \omega_i) = \frac{k_i}{nV} \tag{2.10}$$

where k_i is the number of points in V that belong to class ω_i. Thus, the posteriori probability is given by

$$p(\omega_i|x) = \frac{p(x, \omega_i)}{\sum_{j=1}^{M} p(x, \omega_j)} = \frac{\frac{k_i}{nV}}{\frac{k}{nV}} = \frac{k_i}{k} \tag{2.11}$$

Thus, the most well represented class among the k nearest neighbors will maximize the posteriori probability.

Support Vector Machines

Assuming a two class classification problem (ω_1 and ω_2), support vector machines are a type of linear discriminant function that attempt to find the hyperplane that separates the classes with the maximum distance between itself and any points in either class (Duda et al., 2000) as shown in Figure 2.2. A linear discriminant function, $g(\mathbf{x}) = w_0 + \sum_{i=1}^{d} w_i x_i$, attempts to find a hyperplane defined by $w_i,\ i = 0, \ldots, d$ such that $g(\mathbf{x}) > 0$ if $\mathbf{x} \in \omega_1$ and $g(\mathbf{x}) < 0$ if $\mathbf{x} \in \omega_2$. This assumes that the classes are linearly separable. If the classes are not linearly separable, then the data points $\mathbf{x}_k$ should be mapped to a higher dimensional space using an transform $\phi(\cdot)$ such that $\mathbf{y}_k = \phi(\mathbf{x}_k)$. Assuming the scalar $z_k = \pm 1$ indicates the class of the data point $\mathbf{y}_k$, a linear discriminant function is given by

$$g(\mathbf{y}) = \mathbf{a}^t \mathbf{y} \tag{2.12}$$

where both the weight vector and transformed vector have been augmented to accommodate a bias weight ($a_0 = w_0$ and $y_0 = 1$). This means a hyperplane that separates the classes must satisfy

$$z_k g(\mathbf{y}_k) \geq 1,\ k = 1, \ldots, n \tag{2.13}$$

However, the goal of SVMs is to maximize the margins between itself and the two classes. Since the distance from the hyperplane to the transformed pattern is given by $\dfrac{|g(\mathbf{y})|}{\parallel \mathbf{a} \parallel}$ and it is assumed a positive margin b exists

$$\frac{z_k g(\mathbf{y}_k)}{\parallel \mathbf{a} \parallel} \geq b,\ k = 1, \ldots, n \tag{2.14}$$

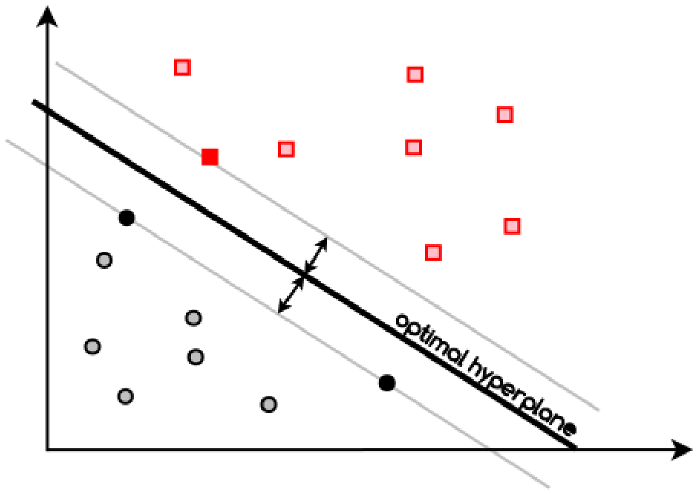


Figure 2.2: A support vector machine attempts to find the hyperplane that maximizes the distance (arrows in figure) between itself and the nearest training samples. The support vectors are the samples nearest to the hyperplane and are shown as solid circles and squares in this figure.

Thus, the goal is to find a weight vector $\mathbf{a}$ that maximizes b. Thus, by using the optimization method of Lagrange undetermined multipliers, the goal becomes to minimize

$$L(\mathbf{a}, \boldsymbol{\alpha}) = \frac{1}{2} \parallel \mathbf{a} \parallel^2 - \sum_{k=1}^{n} \alpha_k [z_k \mathbf{a}^t \mathbf{y}_k - 1] \tag{2.15}$$

with respect to $\mathbf{a}$ and maximize it with respect to the undetermined multipliers $\alpha_k \geq 0$. This can be reformulated as maximizing

$$L(\boldsymbol{\alpha}) = \sum_{k=1}^{n} \alpha_k - \frac{1}{2} \sum_{k,j} \alpha_k \alpha_j z_k z_j \mathbf{y}_j^t \mathbf{y}_k \tag{2.16}$$

subject to the constraints

$$\sum_{k=1}^{n} z_k \alpha_k = 0, \ \alpha_k \geq 0, \ k = 1, \ldots, n \tag{2.17}$$

Neural Networks

As described in the previous section, linear discriminant functions can be used on non-linearly separable problems by using a nonlinear transform function $\mathbf{y}_k = \phi(\mathbf{x}_k)$ that maps the problem to a linearly separable space. However, determining what this transform should be relies on having some knowledge relevant to the problem. Multi-layer feed-forward neural networks approach this problem by learning the non-linear mapping at the same time they learn the linear discriminant function (Duda et al., 2000).

The discussion in this section will be limited to single hidden layer neural networks that take the form

$$g_k(\mathbf{x}) \equiv z_k = f_O\left(\sum_{j=1}^{n_H} w_{kj}\; f_H\left(\sum_{i=1}^{d} w_{ji}x_i + w_{j0}\right) + w_{k0}\right) \tag{2.18}$$

where z_k refers to the k^{th} output of the neural network, f_O and F_H are the activation functions of the output layer and hidden layer respectively, n_H is the number of neurons in the hidden layer, d is the dimensionality of the input vector, w_{kj} and w_{ji} refer to the weights of the output and input layer respectively, and w_{k0} and w_{j0} refer to the bias weights. It can easily be seen that this is simply two linear discriminant functions with an activation function, almost always non-linear, separating them.

Feed forward neural networks are frequently trained using the backpropagation method that is based on gradient descent. The training error for a neural network with c output neurons is defined as

$$J(\mathbf{w}) \equiv \frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2 = \frac{1}{2} \parallel \mathbf{t} - \mathbf{z} \parallel^2 \tag{2.19}$$

where $\mathbf{t}$ and $\mathbf{z}$ are the target and output vectors respectively. The weights of the network are initialized randomly and then updated in a direction that reduces the error using gradient descent

$$\Delta w_{pq} = -\eta \frac{\partial J}{\partial w_{pq}} \tag{2.20}$$

where η is a the learning rate that regulates the size of the change in weights. From the equations given the following equations can be derived that allows the neural network in Equation 2.18 to be trained. First, it is denoted that:

$$net_j = \sum_{i=1}^{d} w_{ji}x_i + w_{j0} \equiv \mathbf{w}_j^t\mathbf{x} \tag{2.21}$$

and

$$net_k = \sum_{h=1}^{n_H} w_{kj}y_j + w_{k0} \equiv \mathbf{w}_k^t\mathbf{y} \tag{2.22}$$

Using this notation it can be shown that the output layer weights should be updated as

$$\Delta w_{kj} = \eta(t_k - z_k)f'(net_k)y_j \tag{2.23}$$

and the hidden layer weights should be updated as

$$\Delta w_{ji} = \eta \left[\sum_{k=1}^{c} w_{kj}(t_k - z_k)f'(net_k)\right] f'(net_j)x_i \tag{2.24}$$

Neural networks are often trained until they begin to overfit the data. This is generally performed by using a labeled validation set to detect when the problem is no longer generalizing the problem well. The number of hidden neurons n_H is typically chosen heuristically based on knowledge of the problem at hand, and the number of output neurons c typically corresponds to the number of classes in a classification problem.

2.1.3 Semi-Supervised Learning

Semi-supervised learning is a subset of supervised learning methods that use both labeled and unlabeled data for training. These methods are particularly useful for tasks where obtaining labeled data is expensive but unlabeled data is plentiful. Semi-supervised learning methods rely on assumptions about the nature of the dataset that is being used (Chapelle et al., 2006). These assumptions include the smoothness assumption, the cluster assumption, and the manifold assumption. These assumptions depend on the idea that points that lie near each other, in the same cluster, or on the same manifold share the same label. If the data does not meet one of these assumptions, then semi-supervised learning methods will not be able to improve over purely supervised methods. Semi-supervised methods leverage these assumptions in order to form a model that increases performance on the learning task.

Two-step learning algorithms are closely related to semi-supervised learning (Chapelle et al., 2006). These perform an unsupervised step on all data such as a change of representation or construction of a new kernel. Then, supervised learning is uses this new representation of the data. DeSTIN is closely related to this class of algorithm since it performs a transformation of the data into a new feature space before utilizing tradition supervised classifiers.

2.2 Feature Extraction

Feature extraction is a process whose goal is to make classification an easier task. An ideal feature extractor would only require a simple classifier, while an ideal classifier would need no feature extractor at all (Duda et al., 2000). A feature can be thought of as a measurement of the characteristics of the data. It is important that the characteristics the features measure aid in differentiating between classes, or performance will suffer. Early development of feature extractors consisted of hand-engineered processes, such as locating points of interest in an image and measuring the distance between them (Gonzalez and Woods, 1992). Much of the literature that exists in the field of feature extraction revolves around domain-specific methods such as feature extractors that work with human faces (Yuille et al., 1992), Arabic handwritten digits (Abdelazeem, 2009), fingerprints (Ratha et al., 1995), and tumor diagnosis (Street et al., 1993). However, the methods developed for these very specific domains are often not helpful in others. Jiang (2009) defines four categories of feature extraction for images: human expert knowledge based methods, image local structure based approaches, image global structure based techniques and machine learning based statistical approaches (Jiang, 2009). The first type, human expert knowledge based methods, typically have no application outside the domain they were designed for. The second type, local image structure approaches, include methods designed to locate edges, lines, ridges, and similar characteristics while the third type, global image structure approaches, includes methods such as histograms and Fourier transforms. The final type, machine learning based statistical approaches, are the most general and have the broadest application ability, and this type of approach is the kind the work presented here can be categorized as. The remainder of this section will outline a couple of machine learning type feature extraction processes.

2.2.1 Principal Component Analysis

Principal component analysis, first introduced by (Pearson, 1901), is often used to map a dataset $\mathbf{X}$ of dimensionality D to a new feature set $\mathbf{T}_L$ of dimensionality $L \leq D$. The first dimension of $\mathbf{T}_L$ will correspond the the direction of greatest variation in the dataset $\mathbf{X}$, while the second dimension will correspond the direction orthogonal to the first dimension with the greatest variance. This process continues until all the dimensions of $\mathbf{T}_L$ are constructed in order of rank by direction of maximum variation. Singular value decomposition (SVD) can be used to produce $\mathbf{T}_L$ (Gerbrands, 1981). In order to retrieve the directions of greatest variation rather than just the directions with the greatest average magnitude, a dataset $\hat{\mathbf{X}}$ must be created from $\mathbf{X}$ such that the mean of each dimension of $\hat{\mathbf{X}}$ is zero. The SVD of $\hat{\mathbf{X}}$ is given by:

$$\hat{\mathbf{X}} = \mathbf{U}\mathbf{S}\mathbf{V}^T \tag{2.25}$$

then $\mathbf{T}_D$ can be calculated as

$$\mathbf{T}_D = \mathbf{US} \tag{2.26}$$

$\mathbf{T}_L$ is then simply the first L columns of $\mathbf{T}_D$. In this wasy, PCA attempts to map the data onto a smaller space while preserving as much information about the data as possible. Assuming that the L directions of greatest variation hold most of the information about the problem, this is true. There is a tradeoff between the reduction in complexity and the loss of information caused by reducing L.

2.2.2 Auto-encoders

Auto-encoders, or auto-associators, are neural networks that are trained with targets that are the same as the input (Bengio, 2009). It has been shown that if the number of hidden units n_H is less than the dimensionality D of the input and the hidden layer has a linear activation function, then the hidden layer learns to project the input onto the span of the first n_H principal components (Bourlard and Kamp, 1988). This makes sense since the output of the hidden layer is a projection onto an n_H dimensional space from which the output layer is trying to minimize the reconstruction error. This error can only be minimized, when the hidden layer is capturing the directions of maximum variation. Thus, auto-encoders can be used as a feature extractor. Although it may seem that auto-encoders with $n_H \geq D$ would simply learn some meaningless projection, such as learning an identity weight matrix, it has been shown in practice that they can yield useful representations for classification (Bengio and LeCun, 2007).

2.3 Image Whitening

The work presented later will include classifying natural images. In order to accomplish this, it is important to whiten the image patches. Whitening is a process whereby the pixels that make up an image are normalized such that they have unit variance and are uncorrelated (Hyvrinen et al., 2009). In order to accomplish this, principal component analysis (PCA) can be utilized.

First, the image patches are normalized to have zero mean. Then the covariance matrix, $\mathbf{\Sigma}$, of the image patches is calculated. SVD is then performed on this matrix. A whitened image patch $\mathbf{X}_{PCA}$ can then by calculated from an image patch $\mathbf{X}$ as shown in Equation 2.27. ϵ is a small constant used to guarantee numeric stability.

$$\mathbf{X}_{PCA} = (\mathbf{S} + \epsilon)^{-\frac{1}{2}}\mathbf{U}^T\mathbf{X} \tag{2.27}$$

However, this whitening process can be further improved. Zero-phase component analysis (ZCA) (Bell and Sejnowski, 1996) whitens the data in such a manner as to minimize the squared error between the original image and the whitened image. This means the whitened image preserves the spatial structure of the original image. A ZCA whitened image patch $\mathbf{X}_{ZCA}$ can then be calculated as given in Equation 2.28.

$$\mathbf{X}_{ZCA} = \mathbf{U}(\mathbf{S} + \epsilon)^{-\frac{1}{2}}\mathbf{U}^T\mathbf{X} \tag{2.28}$$

2.4 Deep Machine Learning

In Section 1.1 the motivation behind DML methods was discussed. In this section a couple of well-known DML methods will be described. Deep Belief Networks (DBN) (Hinton et al., 2006) and Convolutional Neural Networks (CNN) (Lee et al., 2009a) are two of the mainstream DML paradigms that have been successfully demonstrated in addressing pattern recognition problems in high dimensional data (e.g. images) (Arel et al., 2010).

2.4.1 Convolutional Neural Networks

Although deep (many layered) neural networks have been found difficult to train, Convolutional Neural Networks (CNNs) have been an exception (Bengio, 2009).

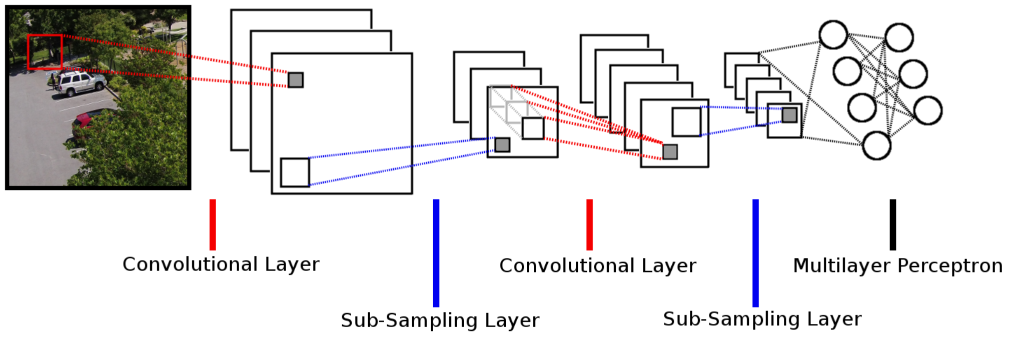


Figure 2.3: Convolutional Neural Network: This figure is depicts a CNN with two convolutional layers and two sub-sampling layers. This example has 3 hidden units in the first convolutional layer and 5 hidden units in the second convolutional layer.

Inspired by the human visual system structure, they are based on the local connectivities between hierarchically organized transformations of the input image. Because of this nature of this system, it is designed for data that can meaningfully be structured into a two-dimensional grid such as an images. CNNs take advantage of three architectural ideas in order to attain some degree of shift, scale, and distortion invariance: local receptive fields, shared weights (or weight replication), and spatial subsampling (LeCun et al., 1998). These methods also enforce sparsity in the architecture and small-fan in at each neuron which has been hypothesized to aid in credit assignment in such a deep neural network (Bengio, 2009).

The CNN process can be summarized as follows. The input image is convolved with a set of N hidden units. Each hidden unit has a $d \times d$ receptive field. Thus, all positions in the convolution share a common set of hidden units. The output of this convolutional layer is N-channel feature map. This process is equivalent to a convolution of N filters with the input image. This layer is then followed by a subsampling (or pooling) layer. This layer performs a subsampling process in order to reduce the dimensionality of the feature map output of the convolutional layer. This is typically performed with a $2x2$ averaging or max operation. This also increases the shift, rotation, and distortion invariance of the system. Then, successive convolutional and subsampling layers alternate until finally the output of the final subsampling layer is used as the input to a locally connected layer. A locally connected layer is functionally identical to a convolutional layer except each position in the image has a unique set of hidden units. Locally connected layers are not followed with subsampling layers since that would entail pooling feature map values from hidden units with different learned parameters. The final locally connected layer is then connected to a fully connected MLP. Convolutional Neural Nets currently have the best performance results on the MNIST handwritten digit dataset (Lecun and Cortes, 2009) and the CIFAR-10 object classification dataset (Krizhevsky and Hinton, 2009).

2.4.2 Deep Belief Networks

Deep Belief Networks (DBNs) are probabilistic generative models which separates them greatly from traditional neural nets which are discriminative in nature (Hinton et al., 2006). Generative models estimate a joint probability distribution over observable data and labels, facilitating the estimation of both $P(Obs.|Label)$ as well as $P(Label|Obs.)$, while discriminative models can only estimate the posterior probability $P(Label|Obs.)$. This means that DBNs can not only predict a label given an observation, but it can predict an observation given a label. Thus, if a DBN is trained to recognize handwritten digits, that DBN can generate a representative image of a '7' if the label '7' is provided to the network.

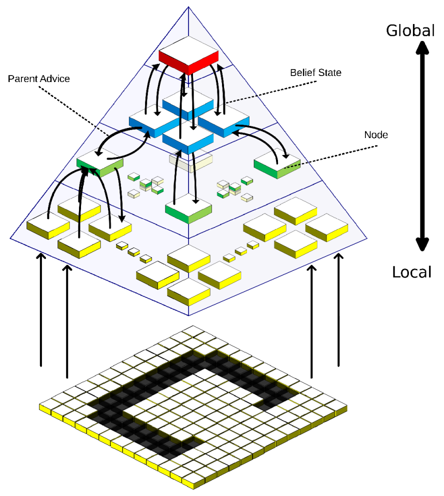
DBNs were designed to avoid the need for a substantial labeled dataset, the slow learning caused by backpropagation, and avoid poor parameter initialization that leads to the system being stuck in poor local optima. Although there has been some success in training DBNs in a completely unsupervised manner (Lee et al., 2011), most current work also performs supervised training after this unsupervised step. DBNs also are not designed to naturally represent temporal information, although there has been some similar work in which a type of RBM designed to capture temporal features have been arranged in a stack in order to naturally learn sequences (Lockett and Miikkulainen, 2009). The most common application for DBNs is static images and DBNs have been competitive on the MNIST benchmark (MarcAurelio Ranzato et al., 2007; Lee et al., 2009b).

DBNs consist of several layers of Restricted Boltzmann Machines (RBMs) which are *restricted* to a single visible layer and a single hidden layer where units within a layer are not connected to one another. The hidden units are trained to capture correlations observed at the visible units. During the initial training phase, these layers are connected only by top-down generative weights. The ease of learning these weights as compared to more traditional neural networks is what makes RBMs more attractive as building blocks for deep layered networks. In order to learn these generative weights, the initial training consists of an unsupervised, greedy layer-by-layer method called contrastive divergence (Hinton, 2002). To summarize this process, a vector $\hat{v}$ is presented to the visible units of the RBMs. The visible unit inputs are then stochastically found going backwards through the RBM in such that the original input is reconstructed. Then, the resulting visible neuron activations are forwarded in order for one step reconstruction hidden unit activations $\hat{v}$ to be obtained. These forward and backward process is called as Gibbs sampling. The resulting weight update is based on the difference in the correlation of the hidden activations and the visible inputs (Arel et al., 2010). The top two layers of the network form an associative memory. The weights of these layers are tied together in order for the lower layer output to provide a reference clue the top layer can associate with its memory contents. DBNs can be further trained after this unsupervised process using labels and backpropagation to improve results. Finally, a set of labels is connected to the associative memory and the bottom-up recognition weights (previously unconnected during the unsupervised pre-training) are learned.

Chapter 3

The DeSTIN Architecture

This chapter describes the Deep Spatio-temporal Inference Network (DeSTIN) architecture, first introduced in Arel et al. (2009), and details efforts to improve its performance and scalability. DeSTIN offers some advantages over other deep learning methods. Compared to the most successful DL method, convolutional neural networks, it offers three primary advantages: faster convergence, more feasible implementation in alternate technologies such as analog circuitry, and the ability to naturally capture spatio-temporal features. DL architectures often contain millions of learned parameters and multiple layers. Back-propagating an error through 5+ layers and millions of parameters in order to correctly adjust each of those parameters takes many iterations over a dataset. DeSTIN uses an incremental clustering algorithm as its basic building block which is able to converge much quicker. The winner take all clustering algorithm used does not have the credit assignment problem of back-propagation since one input only causes one centroid to be updated and thus takes many fewer iterations to converge. DeSTIN is also a more feasible architecture to implement in analog circuitry. Implementing an architecture that consists of series of clustering nodes sparsely connected together like DeSTIN is more feasibility than CNN's which contain many types of layers, large feature maps between layers, and much more dense connections between layers. This increased feasibility comes from



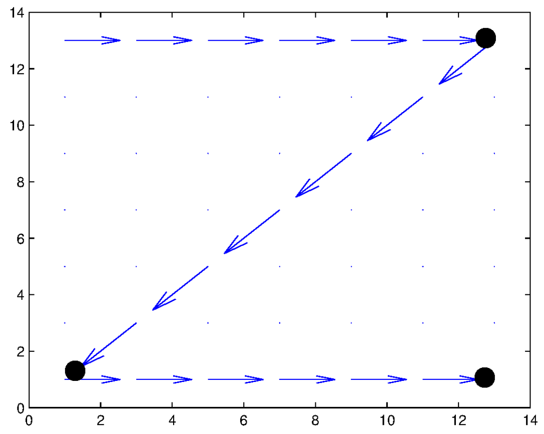
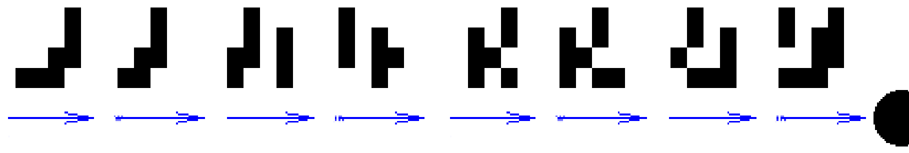



Figure 3.1: Typical use of DeSTIN in an image classification task. (TOP) The images are provided to the bottom layer nodes. (MIDDLE) The bottom layer viewing window is scanned over the image in a specific pattern. Belief states from specific movements are saved as indicated by the black dots. (BOTTOM) The belief states must represent the sequence of inputs provided to each node.

less design work (one type of layer), less connectivity between layers, and decreased memory requirements on the chip (no feature maps between layers).

DeSTIN consists of multiple instantiations of an identical functional unit called a node which learns to generate features via a completely unsupervised learning process, unlike most of the previously discussed DML methods that rely on labeled information. These nodes are arranged in layers, where each node is assigned children nodes from the layer below and a parent node from the layer above as shown in Figure 3.1. Nodes at the lowest layer receive as input a subset of the raw sensory data while nodes at all other layers receive the belief states, or outputs, from their children nodes as input. This subset, or receptive field, will be unique for each node. The receptive fields are disjoint in the work presented here, but could be overlapping depending on the application. Each node attempts to capture the salient spatio-temporal regularities contained in its input and continuously update a belief state meant to characterize the input and the sequences thereof. The belief state (or belief) is a probability vector that indicates the probability of each possible state given the information we know about the system. The beliefs formed throughout the architecture can then be used as rich features for a classifier that can be trained using supervised learning. Beliefs extracted from the lower layers will characterize local features and beliefs from higher layers will characterize global features. Thus, DeSTIN can be viewed as an unsupervised feature extraction engine that forms features from data based on regularities it observes as shown in Figure 3.2. This stands in contrast to the user-engineered features based approach, which relies on previous knowledge of the problem at hand. The unsupervised nature of DeSTIN renders it much simpler to train than other DML architectures (Karnowski, 2012). It also suggests that the features generated by DeSTIN need not be unique in the sense that they converge to singular values in order to be meaningful to a given application.

As outlined above, the core function of each node is to form a belief state that characterizes the inputs observed. This belief state is expressed through the following

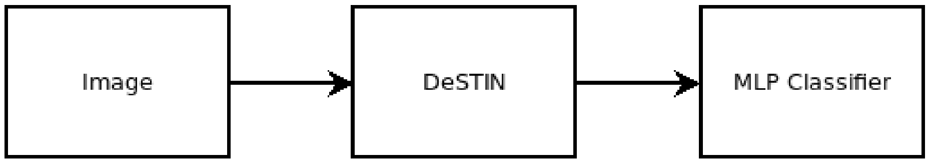


Figure 3.2: Typical use of DeSTIN in an image classification task. The images are provided to DeSTIN which generates features that are supplied to a classifier.

conditional probability function

$$b_t(s_t|a) = \frac{\Pr(o|s_t)\left\{\sum_{s_t \in S} \Pr(s_t|s_{t-1}, a_{t-1}) b(s_{t-1})\right\}}{\sum_{s'_t \in S}\left\{\Pr(o|s'_t)\sum_{s''_t \in S}\Pr(s''_t|s_{t-1}, a) b(s_{t-1})\right\}} \tag{3.1}$$

which serves as an update equation, as the system transitions from one time step to the next. This function maps the input o from the layer below, belief state b (which is a function of the system state s), and parent's belief state(i.e. advice) a from the layer above to an updated belief state $b_t(s_t)$. The denominator of this equation is a normalization factor. This equation should be viewed as two parts: (1) a posterior over the observations, $\Pr(o|s_t)$, that is modulated by a (2) construct that reflects the system dynamics, $\sum_{s_t \in S} \Pr(s_t|s_{t-1}, a_{t-1}) b(s_{t-1})$. These building blocks of the architecture are the pieces of information which must be learned from the data. Figure 3.3 demonstrates how this temporal mechanism is used to capture information as DeSTIN's viewing window is scanned across an image.

3.1 Incremental Clustering

Since DeSTIN was designed as a system that is scalable using simple hardware, an incremental clustering algorithm is employed for learning $\Pr(o|s_t)$ in order to minimize memory requirements. Young et al. (2010) introduced the winner-take-all incremental clustering algorithm used as the core of each DeSTIN node, though several modifications have been made to the clustering algorithm itself in order to

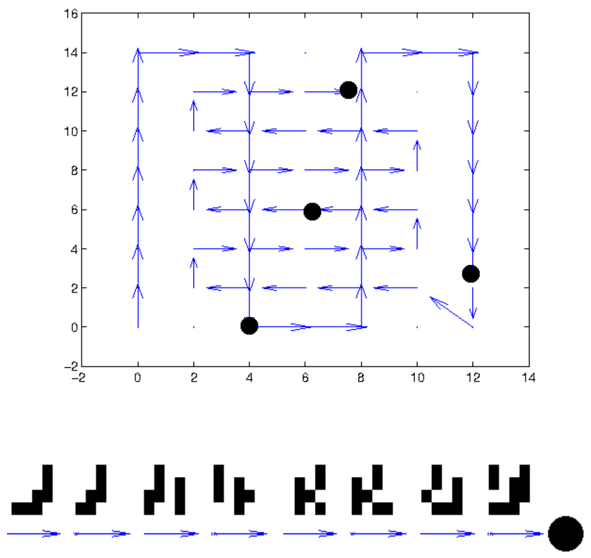


Figure 3.3: Typical use of DeSTIN in an image classification task. DeSTIN's receptive field (bottom layer inputs) is scanned across the image. Belief states from specific movements are saved to be used as features to a classifier. Thus the belief states from a specific movement must characterize the inputs seen before that movement.

eliminate unnecessary computation and help generate richer features. The algorithm finds centroids which are represented by a mean μ and variance σ^2 in each dimension. Based on the centroids formed and their relationship to the input vector o, $\Pr(o|s_t)$ is obtained where s_t corresponds to a particular centroid in the set of centroids. A key idea of this algorithm was the introduction of the starvation trace which addressed centroids that happen to be initialized far from any dense regions of the observation space and thus would never be selected for update. The starvation trace is used to shrink the apparent distance of a starved centroid to all input vectors until it is selected for update. A starvation trace value, ψ, is maintained for each centroid and is decayed by a constant, γ, each time that centroid is not updated and increases once the centroid is selected, as reflected by

$$\psi_c = \gamma\psi_c + (1-\gamma)\mathbb{1}_{x=c} \tag{3.2}$$

where x represents the chosen centroid. Starvation trace is utilized to weigh the distances used to select the centroid to be updated, such that

$$x = \text{argmin}_{c \in C} \left[\psi_c \left\| o - \mu_c \right\| \right] \tag{3.3}$$

where x is the centroid to be updated and C is the set of all centroids.

When a centroid is selected for an update, its mean is updated in the direction of the input vector and its variance estimate is updated as follows:

$$\mu_x = \mu_x + \alpha(o - \mu_x) \tag{3.4}$$

$$\sigma_x^2 = \sigma_x^2 + \beta \left[(o - \mu_x)^2 - \sigma_x^2 \right] \tag{3.5}$$

Upon updating the selected centroid, the posterior distribution, $\Pr(o|s')$, is obtained using the normalized Euclidean distance between the input and each centroid c, such that

$$n_c = \sum_{i=1}^{d} \frac{(o_i - \mu_{c,i})^2}{\sigma_{c,i}^2} \tag{3.6}$$

$$p_c = \frac{n_c^{-1}}{\sum\limits_{c' \in C} n_{c'}^{-1}} \tag{3.7}$$

where p_c represents the probability the observation belongs to the centroid c.

This is a departure from previous work (Karnowski, 2012; Karnowski et al., 2010), where the posterior distribution was calculated as either a simple function of the Euclidean distance or by sampling an exponential probability density function centered at the centroid mean. The former method is lacking because it does not take into account the variance of the data a centroid represents. The latter method is lacking because it tends to form unreasonably confident beliefs for input vectors that are not near any centroid. It also complicates the calculation without adding any more information content to the belief construct.

3.2 The DeSTIN Node Revisited

Here, several changes to previous implementations of DeSTIN are presented. Previously, each node was performing several dissimilar tasks in order to model the system dynamics. The memory and/or computation requirements of these methods dwarfed the resource requirements of the clustering algorithm that is supposed to be the core of each DeSTIN node. Even without the resource requirements imposed by these methods, the fact that there are many dissimilar operations required renders mapping the architecture to a parallel platform rather challenging. The changes presented here aim to include this functionality into the core clustering algorithm in order to lessen these resource requirements and make the process of implementing DeSTIN in parallel platforms a more attainable task.

3.2.1 Recurrent Clustering

In order to more easily map the DeSTIN architecture to a parallel implementation, the mechanism used by the original DeSTIN architecture to capture temporal information, reflected by the construct $\Pr(s_t|s_{t-1}, a_{t-1})$, needed to be revised. The philosophical approach taken was to integrate the feedback/recurrence mechanism as an inherent part of the clustering process. In previous work, the temporal regularities $\Pr(s_t|s_{t-1}, a_{t-1})$ were captured by maintaining a table populated with the likelihoods of transitioning between states or through a function approximation method that attempted to predict the next state given the current state. Though keeping a table of transition probabilities seems simple enough, the manner in which it was being used necessitated that an array be kept for every movement made across the image and for every possible belief state provided by the parent node. This results in a

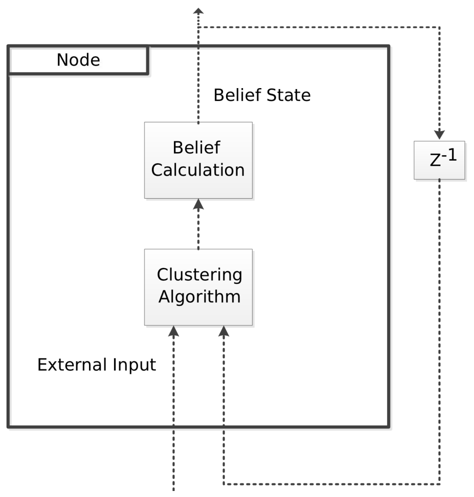


Figure 3.4: In recurrent clustering the previous belief is latched and augmented to the input over which clustering is performed.

table that has a memory requirement for a single node of $M_{tab} = K^2AL$, where K is the number of centroids for the node, L is the number of movements, and A is the number of belief states from the parent. In addition to the large memory requirement, using this table mandated an additional set of operations outside of the core node functionalities of clustering and calculating $\Pr(o|s_t)$. The other previously used mechanism to estimate $\Pr(s_t|s_{t-1}, a_{t-1})$ is function approximation (Karnowski et al., 2011). While this method has a more modest memory footprint, it requires an extensive set of operations outside the core functionality of the node. These additional operations make it incredibly difficult to map the DeSTIN architecture to a parallel platform like a GPU or custom analog circuitry. For this reason, it is desired to couple the learning of temporal regularities and parent advice more closely with the clustering mechanism.

To address this problem, recurrent clustering is proposed as illustrated in Figure 3.4. Recurrent clustering takes as input the external input augmented by the node's previous time step belief. This allows the clustering to form beliefs that are based on both spatial and temporal attributes. Consequently, μ and σ^2 have dimensions $K \times (N + K)$, where K is the number of centroids and N is the number of input dimensions. It is important to note that in addition to allowing the clustering mechanism to capture temporal dependencies, this method facilitates the formation of centroids that represent relationships between the spatial and temporal features of the data. During the clustering process, the centroids will converge to values that represent spatial and temporal regularities in the data. Previously, the clustering algorithm could only observe the input vector and characterize it's similarity to other input vectors. In this revised formulation, the clustering results in belief states that represent information about transitions between belief states or, more generally, about a sequence of transitions between belief states, since each belief depends on the preceding belief.

There are many hazards to consider in designing the recurrent clustering mechanism due mainly to the introduction of a feedback loop. The most important aspect to consider is the method used for determining which centroid is to be updated, and the derivation of its respective belief state. It is imperative that the clusters formed characterize both the temporal and spatial attributes of the data. As a result, it is key to balance the contributions of the spatial and temporal components of the input structure when selecting the centroid to update. This means that a selection method that uses the centroid variances to weight the importance of each dimension can not be used because it encourages the recurrent clustering algorithm to form very confident beliefs that only consider temporal attributes. The result is a system that can only act as a counter and provides no information about the inputs it observes. For this reason, the selection method used is based solely on the Euclidean distance between the centroid means and the combined input/belief vector, as suggested by the selection rule of eq. (3.3). If the variances in the beliefs are expected to be much different than the variances in the input data, it might be necessary to use a distance measure that balances the spatial and temporal dimensions in order to prevent either from improperly dominating the clustering process. However, this has not been necessary in DeSTIN or any other applications explored thus far. Once the

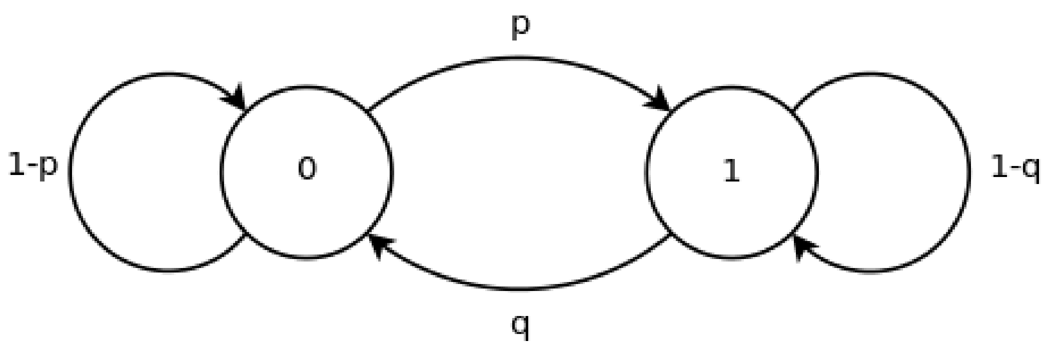


Figure 3.5: A two-state Markov Chain

winning centroid has been updated, the belief state is calculated as outlined in eq. (3.6) and (3.7).

3.2.2 Recurrent Clustering Models a Markov Chain

In this section, the properties of recurrent clustering that allow it to model a Markov chain are explained. For the purposes used here, the extrensic input will refer to the value that a state in the Markov Chain emits. This value will be unique for each state.

Definition 3.1. *(Centroids). A clustering model is defined by a set of k centroids. The centroids means, $\mu_1 \ldots \mu_k$, define the center of the data clusters.*

Definition 3.2. *(Input Vectors). Each input vector, $X_{i,n}$, comprises elements originating from an external source (i.e extrensic input) and elements of the previous belief state (i.e. intrinsic input). $X_{i,n}$ refers to the n^{th} input vector belonging to the i^{th} centroid.*

Definition 3.3. *(Clustering Error). Clustering error, $\epsilon(\mu, X)$, is defined as the sum of the distances between each input vector and its corresponding centroid, $\sum_{i=1}^{k}\left(\sum_{n=1}^{N_i}\|\mu_i - X_{i,n}\|\right)$. Winner-take-all clustering seeks to minimize this error.*

When the centroids are randomly initialized, elements of the input vectors, corresponding to the intrinsic input, will tend to have greater variance as no regularities have yet been discovered. This holds until such time as the extrensic regularities are modeled by the centroids. Hence, during this period, the extrenisic regularities will dominate the clustering algorithm since they have lower variance than that of the intrinsic regularities. This suggests that the clustering error can only be minimized in the space comprised of the extrensic inputs. If the means, M_e, of the extrensic input clusters are known and the elements of $U \in \mathbb{R}^{kxk}$ are all set equal to $1/k$, then the centroids that would minimize the clustering error at $t = 0$ are given by Equation 3.8. The uniform distribution of U is a result of the lack of regularity exhibited by the intrinsic inputs.

$$\hat{\mu} = argmin_{\mu}\ \epsilon(\mu, X) \simeq [M_e\, U]\,,\ t \simeq 0 \tag{3.8}$$

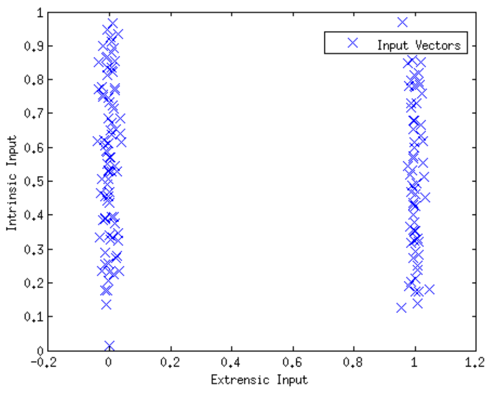
Once the extrensic elements of the centroids are found, the intrinsic dimensions can begin to be learned. The intrinsic elements of the input vector denote the probability that the previous input vector belonged to each centroid. With the extrensic regularities anchoring them, each centroid will minimize its clustering error when its intrinsic dimensions are equal to the expected value of the intrinsic dimensions of its inputs.

$$\hat{\mu}_{i,j} = argmin_{\mu_{i,j}}(\sum_{n=1}^{N} \|\mu_{i,j} - X_{i,n,j}\|)\ 1 \leq i \leq k, j \in intrinsic\ dimension \tag{3.9}$$

$$\hat{\mu}_{i,j} = E[X_{i,1:N_i,j}] \tag{3.10}$$

Thus, the value of a belief element j of a centroid i that will minimize the distance between the centroid and its claimed samples is the probability that the previous state was j given the current state was i,

$$\mu_{i,j} = P(S_{t-1} = j | S_t = i) \tag{3.11}$$



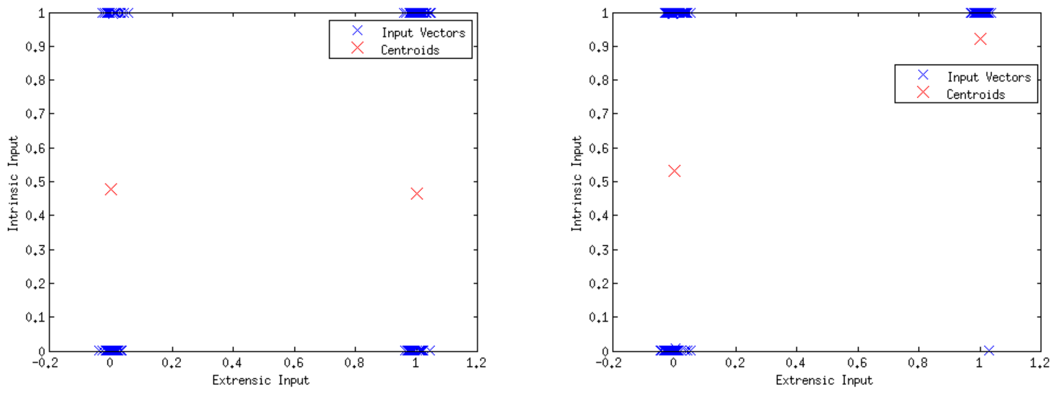


Figure 3.6: (TOP) The distribution of the input vectors at $t = 0$. (LEFT) Distribution and centroids at $t \gg 0$ (p=0.5,q=0.5). (RIGHT) Distribution and centroids at $t \gg 0$ (p=0.5,q=0.9).

The above clearly implies that the centroids represent the states of a Markov chain in the extrensic components and the transition probabilities between these states in the intrinsic components.

3.2.3 Decay of Temporal Information

In this section, the ability of recurrent clustering to capture temporal information will be investigated.

Definition 3.4. *(Belief State). A Belief State is a probability mass function over the likelihood of a sample to belong to each of the centroids. Thus, all samples generating the same belief state will hold a similar distance relationship to each of the centroids.*

Assumption 3.1. *Similar vectors have similar meanings. A basic assumption of clustering is that samples which are spatially close to one another have similar meaning. Thus, any samples belonging to the same cluster should have similar meaning. Furthermore, any samples similar enough to produce the same belief state should share even greater similarity in meaning.*

Axiom 3.1. *If a belief state contains all spatio-temporal information about the system it is attempting to characterize, then it should be able to determine all previous input vectors (extrensic input + previous belief) or input vectors with the same meaning.*

However, from any belief state, the previous input vector cannot be uniquely determined from the current belief state. Only the subset, $\mathbb{A}$, of vectors that could produce that belief state can be determined. If some of these vectors don't have the same meaning, $F(x)$, information about previous states of the system is lost.

Assumption 3.2. *It is assumed that all vectors in this subset have the same meaning as the vector of interest with probability λ.*

$$\mathbb{P}\left(F(a) = F(b)\right) = \lambda \;\; \forall \;\; a, b \in \mathbb{A} \tag{3.12}$$

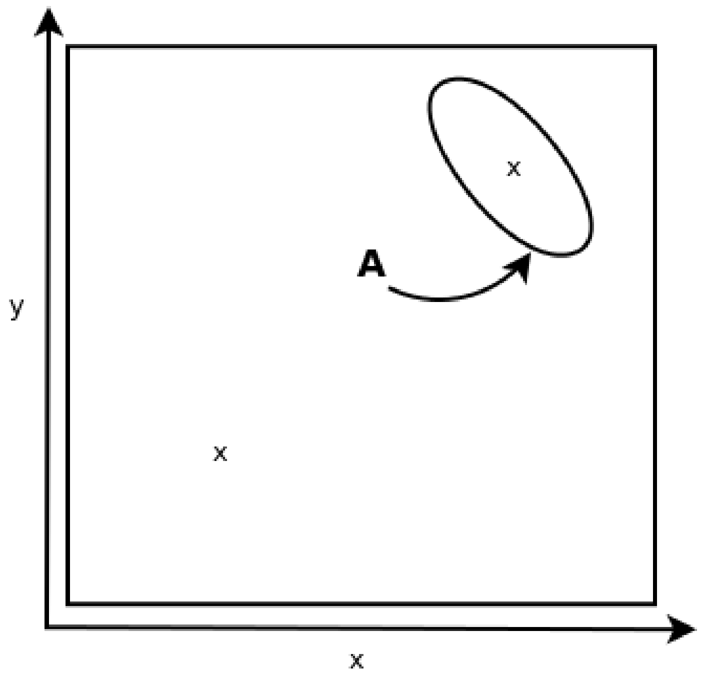


Figure 3.7: Two centroids and a subset of input vectors, $\mathbb{A}$, that generate the same belief state

This implies that with probability λ an input vector with the same meaning as the previous input vector can be determined. Since a portion of this previous input vector is the belief state, this can then be used to attempt to determine its previous input vector. Theorem 3.1 is the result of combining the idea of calculating the input vectors backwards in time with Assumption 3.2, which states that this calculation can only be performed accurately at each time-step with probability λ.

Theorem 3.1. *B_t, the belief state at time t, can be used to determine a vector $\hat{i}_{t-k}$ that contains the same meaning as the input vector i_{t-k} with probability γ, such that*

$$\gamma \propto \lambda^k \tag{3.13}$$

3.2.4 Enhanced Cortical Circuit

Examining eq. (3.1) reveals that the system needs to be able to estimate the probability of the subsequent state given the information received from the parent

Algorithm 2 DeSTIN Pseudocode: This process is performed at every node in the pipelined hierarchy each when an example is presented to the hierarchy.

```
 1: $o \leftarrow [child_1.p_c \; \ldots \; child_N.p_c \; self.p_c \; parent.p_c]$
 2: if TRAINING then
 3:     $x \leftarrow \text{argmin}_{c \in C} \left[\psi_c \|o - \mu_c\|\right]$
 4:     $\mu_x \leftarrow \mu_x + \alpha(o - \mu_x)$
 5:     $\sigma_x^2 \leftarrow \sigma_x^2 + \beta\left[(o - \mu_x)^2 - \sigma_x^2\right]$
 6:     $\psi_c \leftarrow \gamma\psi_c + (1 - \gamma)\mathbb{1}_{x=c}$
 7: end if
 8: $n_c \leftarrow \sum_{i=1}^{d} \frac{(o_i - \mu_{c,i})^2}{\sigma_{c,i}^2}$
 9: {Synchronize Nodes}
10: $p_c \leftarrow \frac{n_c^{-1}}{\sum\limits_{c' \in C} n_{c'}^{-1}}$
```

node. This may be achieved simply by providing the belief state of the parent node as an additional input to the clustering algorithm of the child node, as depicted in Figure 3.8. Thus, parent belief is handled much like the node's own previous belief and hence harmful feedback is avoided using the same mechanisms already employed inside the recurrent clustering algorithm. The system is now able to form beliefs based on local spatial information (the input), local temporal information (the node's previous belief state), and a more global form of advice in the form of the parent node's belief state.

The revised DeSTIN architecture is greatly simplified relative to its predecessor. The memory footprint has been reduced and consolidated into a simple set of two-dimensional matrices. Taking into account the dominating constructs involved, namely the centroid means, variances, starvation traces, and previous belief state, the memory requirement for a single node becomes $M_{node} = 2K(K + N) + 2K$ where K is the number of centroids and N is the number of input dimensions. There are only two core processes taking place at each node and those are very similar and share the same data structure. This reduced architecture, outlined in Algorithm 2, makes implementing DeSTIN on a GPU a far more realistic undertaking. It also suggests that larger topologies, which would be needed for larger problems (e.g. streaming video data), can fit onto a single GPU.

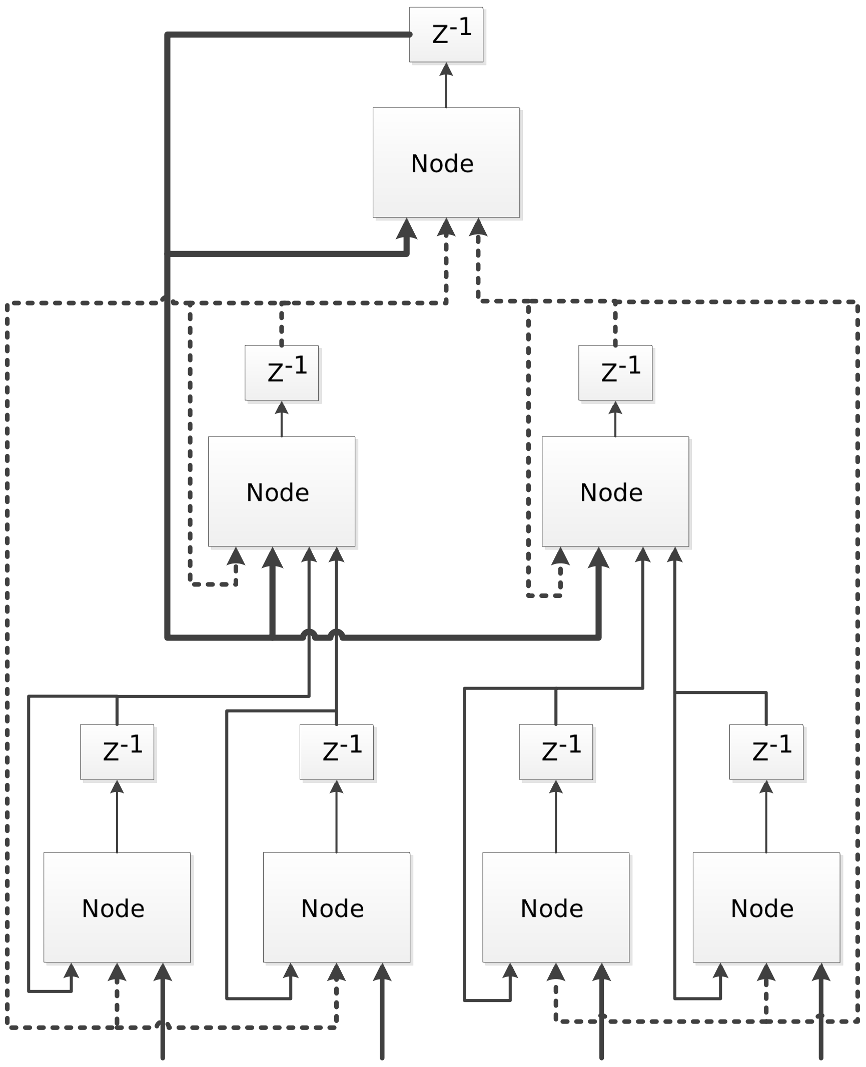


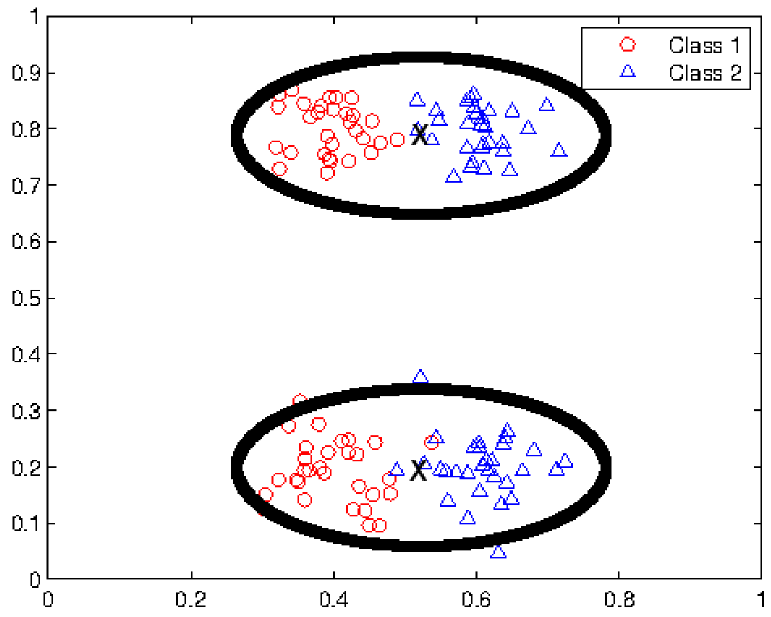
Figure 3.8: A 4-layer DeSTIN architecture illustrating the bottom-up and top-down signaling that is involved. All nodes operate independently and in parallel such that each layer is delayed by one unit of time relative to the layer below it.

3.3 Supervised Clustering

While unsupervised clustering methods strive to capture regularities in a dataset, they do not necessarily capture the most relevant regularities in order to classify the samples. In order to better represent the differences between classes, the class labels can be used to help ensure that the clustering model captures regularities that help discriminate the classes. By appending the label, y, to the input vector, o, a new input vector, $\hat{o} = [o\ y]$, is formed that allows improved centroids to be formed (Chen et al., 1993; Pedrycz, 1998; Uykan et al., 2000). In some cases, the original input vector and the label can have very different scales or distributions and cause an imbalance in the importance attributed to the original input vector, o, and the label, y. In this case, a scaling factor, β, can be used to modify the range of the output variable, $\hat{o} = [o\ \beta y]$ (Chen et al., 1993; Pedrycz, 1998; Uykan et al., 2000). The centroids, $\hat{\mu} = [\mu\ \mu_y]$, that are learned during this supervised training can then be projected onto the space consisting of only the the original input vectors for testing.

Previous work provides little insight into selecting a proper value for β other than to say that it is not overly critical and should be experimentally chosen (Pedrycz, 1998). If the chose value of β is too small, then the labels will not have an effect on the resulting clustering model. A larger β value will result in a clustering model where each cluster is more likely to represent a single class. It is important to note that these more homogeneous clusters do not always provide more information for the purpose of classification. Since the clusters must be projected onto the space that excludes the label, this could result in many identical or greatly overlapping clusters. At best, this results in clusters that add no information. For example, if the model consists of K clusters and two clusters overlap greatly, then $K - 1$ clusters could represent the same model. At worst, these near identical clusters can result in highly different representations of near identical data points.

Since it is obvious that neither $\beta = 0$ nor $\beta = \infty$ are the optimal choice, deciding on an optimal choice of β is a concern. A large enough value should be chosen such



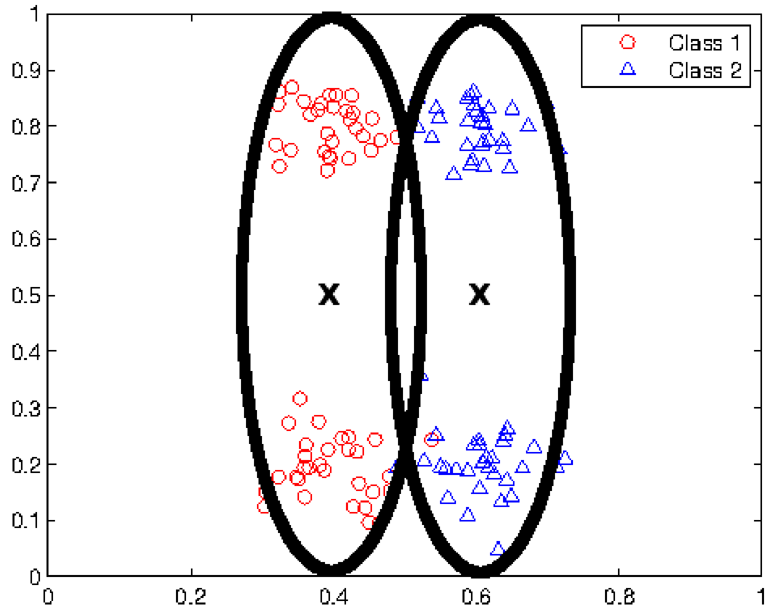
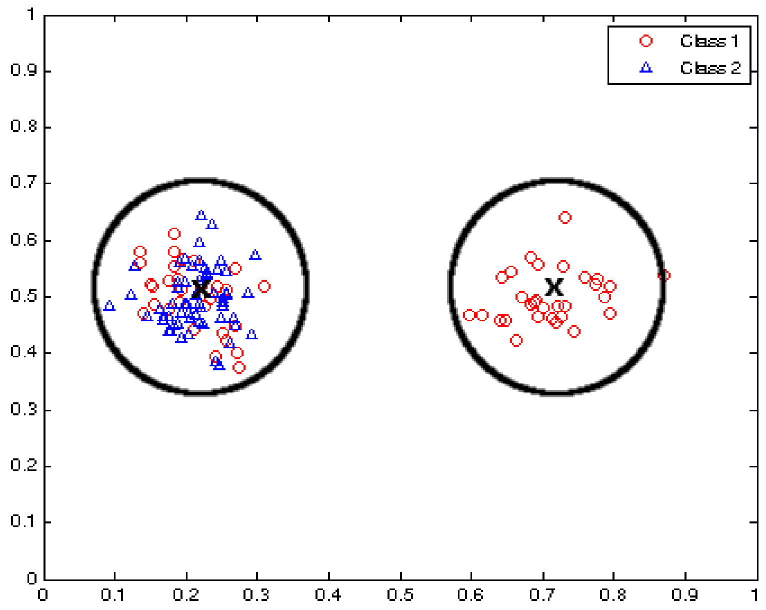


Figure 3.9: A case where supervised clustering improves the clustering model. (Top) Without labels to drive centroids toward intra-class regularities, the resulting model does not separate the classes well. (Bottom) Once labels are used, the centroids are able to discover intra-class regularities and are able to separate the classes well.



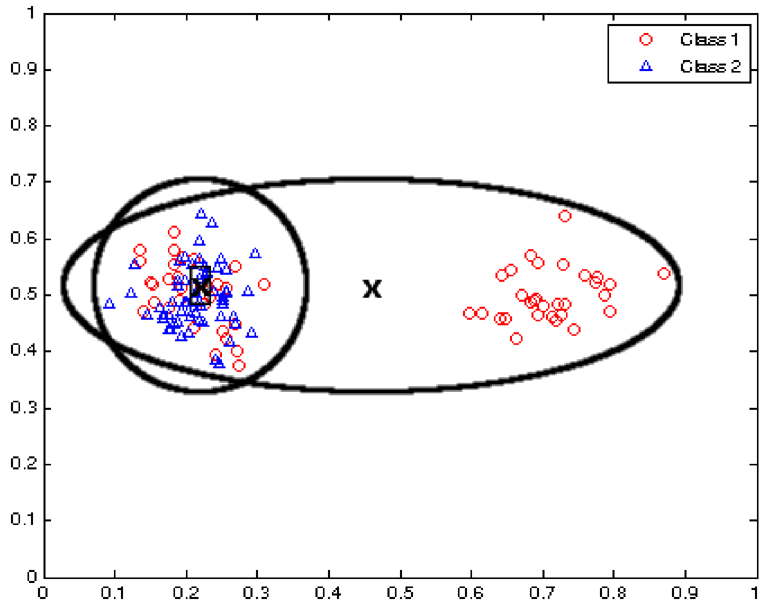


Figure 3.10: A case where supervised clustering degrades the clustering model. (Top) Without labels driving the clustering, the data is well represented and a clustering model is formed that represents the data well. (Bottom) Once labels are introduced, the none of the resulting centroids ends up representing only a single class.

that each cluster is encouraged to represent points from a single class, but the value should be small enough to avoid forming clusters that are near identical in the space that does not include the label. Figure 3.9 demonstrates a case where supervised clustering can improve discrimination between classes, while Figure 3.9 demonstrates a case where it results in identical clusters. It can be observed that when identical clusters are formed the average variance of the centroids that make up the clustering model increases. Thus, a value for β can be chosen by selecting the largest value such that the average variance of the centroids in the clustering model is not increased. In Section 5, experimental results on standard benchmarks indicate that this method of choosing the β results in the the best classification results.

3.4 Scaling DeSTIN to CIFAR-10

The naive approach to applying DeSTIN to larger, more complex problems (e.g. the CIFAR-10 dataset which is widely used as a benchmark in deep learning) is sampling belief states from more DeSTIN nodes and/or movements. Since simply sampling more belief states to produce a larger feature vector does not scale, it was proposed to provide unique sub-samples of this belief data to each classifier in an ensemble of classifiers. However, there is a fundamental reason this approach cannot be successful. Scanning over an image using movements and sampling belief states from positions in this movement sequence does not allow DeSTIN to be invariant to the large shifts in low-level features that exist in CIFAR-10. The following sections discuss why the sub-sampling approach does not address the problem and how the problem can be addressed.

3.4.1 Sampling more movements nodes does not address shifts in features

The above problem of number of large feature vectors, however, was not the barrier to good performance on the CIFAR-10 dataset. The barrier was solving the problem of DeSTIN, as previously used, not being invariant to large shifts in the location of features in the image. In datasets like the MNIST handwritten digit dataset, large shifts in feature locations do not occur. While the segments of the digits might not be located in the exact same location, the expected shifts are small. Thus, as DeSTIN's viewing window is scanned over the image and belief states from specific movements are saved, it is not required that DeSTIN be invariant to these large spatial shifts (or temporal shifts when viewing movements as time) or to the changes in the order that features are scanned. In order to provide DeSTIN with this shift invariance, inspiration is drawn from Convolutional Neural Networks.

3.4.2 Convolutional DeSTIN

Previous work in applying DeSTIN to more complex image datasets such as CIFAR-10 (Krizhevsky and Hinton, 2009) had been unsuccessful. With the MNIST dataset (Lecun and Cortes, 2009), temporality is structured into the movement sequence. This is suitable for MNIST since the location of features in the handwritten digits is subject only to small shifts. However, in datasets like CIFAR-10, the location of features in an image are often much less important than what features occur in the image. Thus, scanning across the image and forcing ”temporality” on the data inhibits classification. Thus, Convolutional DeSTIN is introduced to solve this problem.

In Convolutional DeSTIN, the bottom layer viewing window of DeSTIN is convolved with the input image and the belief states from each quadrant of the image are pooled (averaged) together and provided to an MLP. To more directly compare this method to CNNs, DeSTIN's bottom layer viewing window is convolved with the image to form a set of feature maps. The number of feature maps is equivalent to the

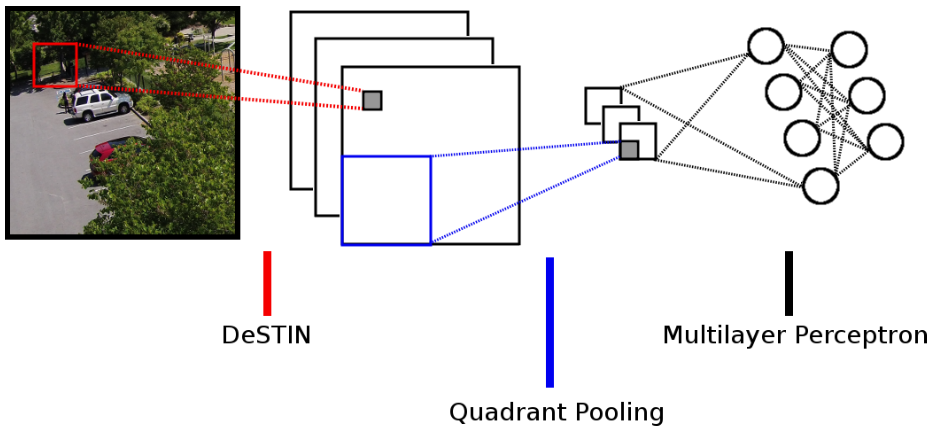


Figure 3.11: Convolutional DeSTIN

number of centroids in the architecture. Sub-sampling by averaging is then performed on each of the quadrants of the resulting feature maps.

Since the higher layers of DeSTIN preserve larger scale features, DeSTIN can take advantage of much larger pooling regions than CNNs. This pooling serves the same purpose as that of CNNs; it provides the architecture with both shift invariance and a reduction in the dimensionality of the features.

Results using this method, without using translations of the training set to increase the number of training examples, are vastly improved over results using the previous ”temporal sampling” method.

3.4.3 Providing Unique Subsamples to Ensemble Members

Previous work (Bryll et al., 2003; Ho, 1998) in providing unique subsets of features to members of an ensemble focuses on the problem of over-fitting when when the number of samples is small compared to the dimensionality of the data. However, my goal for using unique subsets in this work was simply to solve the computational complexity problem of large feature vectors that would be required if more nodes or centroids are used. In the 4:1 child to parent ratio architecture that is often used with DeSTIN, the size of the feature vector would grow by a factor of four with each layer added to the architectures. The resulting high-dimensional belief space causes computational resource problems and over-fitting problems. A better approach in this situation would be to select a subset of the belief states from all layers of the architecture and provide these to the classifier. However, when using Convolutional DeSTIN, feature vectors too large to provide to a single classifier are not a problem.

The classifier used in this research is a negatively correlated ensemble of neural networks (Mishtal and Arel, 2012). With ensemble methods, it is important that the learners do not all produce the same result for each of the inputs. In other words, if every member of the ensemble produces the same output all the time, resources are wasted. One approach to promoting diversity among the learners is negatively correlated learning which enforces diversity by adding an explicit diversity term to the cost function of the neural network learners.

Another way to encourage diversity, while also letting the ensemble work with more information from the DeSTIN hierarchy, is to let each learner work on a unique subset of the belief states. By providing each member of the ensemble with a different subset of the belief states, each learner will naturally form unique hypothesis. Assuming this new source of diversity is strong enough, no communication between the learners is necessary and the training process becomes embarrassingly parallel. Similar methods have been used to not only reduce the dimensionality of the problem each classifier works on for computation purposes, but to boost the overall performance of the ensemble result versus using the entire feature subspace when using simpler classifiers such as decision trees and kNN classifiers (Bryll et al., 2003; Ho, 1998). This result is not expected when using more complex classifiers, but it does speak to the ability of unique subsets to drive diversity in ensembles.

Providing unique subsets of the features to members of an ensemble can certainly drive diversity in the ensemble, but this is not as powerful as using negatively correlated ensemble members (Mishtal and Arel, 2012). Using unique subsets of features was proposed to reduce the number of features provided to the MLP classifier, however is not necessary since it was falsely assumed that saving more movements or adding more nodes to the DeSTIN architecture would be required to apply DeSTIN to the CIFAR-10 dataset. Some results using unique subsets of features generated by DeSTIN with ensembles are provided in the Appendix.

Chapter 4

Implications of Analog Computation Inaccuracies

The possibility of an implementation of DeSTIN in analog circuitry has been an important factor in many design decisions that have been made in this work. In this chapter, the concerns of implementing the core functions of DeSTIN in analog circuitry will be discussed. The primary sources of inaccuracies that occur are mismatch and noise. Mismatch occurs due to random variation in manufacturing circuit components (Tsividis and McAndrew, 2011, sec. 9.7). The manufacturing error that dominates this mismatch error is the variation in the threshold voltages of MOS transistors. When a value is represented with a voltage, known as voltage-mode signaling, this mismatch error manifests as an additive error. However, when current-mode signaling is used it manifests itself as gain error. The standard deviation of these errors is inversely proportional to the area of the transistor (Young et al., 2014b). Thus, reduction a reduction errors results in decreased density and a decrease in computational throughput due to increased capacitance. It is important to note that there is a trade-off between minimizing the mismatch and the power consumed, the physical size of the transistor, and the speed at which calculations can be performed. The errors will not be correlated across the dimensions of the input space or across

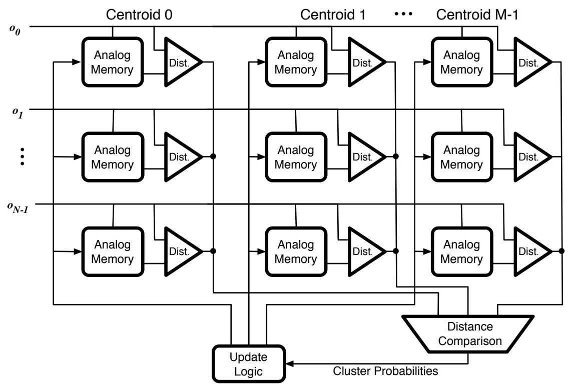


Figure 4.1: This is a block diagram of an analog clustering circuit courtesy of Junjie Lu. o_i refers to the dimension i of the input.

the centroids since the errors are generated by independent transistors. Thus, the error term for each unit may be modeled by an i.i.d random variable.

4.1 Generic Analog Clustering Model

A generic clustering architecture is illustrated in Fig. 4.1, showing an N-dimensional input and M centroids. The functions of our incremental clustering algorithm can be implemented by the following four functions in analog circuitry.

$$d_{i,j} = (o_i - \mu_{i,j})^2 \tag{4.1}$$

$$D_j = \sum_{i=1}^{N} d_{i,j} \tag{4.2}$$

$$\mu_{i,j}[t+1] = \begin{cases} \mu_{i,j}[t] + \alpha(o_i - \mu_{i,j}) & \text{if } D_j = \min(D_{1:M}) \\ \mu_{i,j}[t] & \text{otherwise.} \end{cases} \tag{4.3}$$

$$\sigma^2_{i,j}[t+1] = \begin{cases} \sigma^2_{i,j}[t] + \beta\left(d_{i,j} - \sigma^2_{i,j}[t]\right) & \text{if } D_j = \min(D_{1:M}) \\ \sigma^2_{i,j} & \text{otherwise} \end{cases} \tag{4.4}$$

$$n_j = \sum_{i=1}^{N} \frac{(o_i - \mu_{i,j})^2}{\sigma_{i,j}^2} \tag{4.5}$$

In the following sections the effects of transistor mismatch on the clustering process will be described and a generic analog clustering model based upon the effects that can be expected to be seen in any analog clustering circuit will be constructed.

4.1.1 Input Variation

Each dimension of the input signal must be copied to each of the centroids. In current-mode signaling, this will be accomplished via a current mirror, and the input will be multiplied by a gain error. With voltage signaling, the same input can be physically shared between centroids without explicit copying, and this error source is avoided. This results in modifications to (4.1) and (4.3) such that for a gain variation $d_{i,j} = f(\epsilon_{i,j} o_i, \mu_{i,j})$ and $c_{i,j}[t+1] = g(\mu_{i,j}[t], \epsilon_{i,j} o_i[t])$. The error term $\epsilon_{i,j}$ is the same in both equations, but is different for every centroid and for every dimension within a centroid.

When modeled as a gain error, input variation error will have the effect of giving certain dimensions of the input more or less weight in the distance calculations and in the calculation of the belief state. The impact of this error will be dependent on the characteristics of the true data clusters. If they are well separated, no significant impact should be expected. If they are not, the system could characterize the input in a different manner than expected if each input dimension were given equal weight as demonstrated in Figure 4.2. Gain errors can cause the input to form centroids in a skewed space. This figure demonstrates the effect of a gain error that is too large, $4x$, in the second dimension of each centroid versus a case with no gain error.

4.1.2 Update Asymmetry and Variation

Mismatch in the update mechanisms may result in asymmetric updates, such that increments to a given memory are of a different magnitude than decrements. Also due

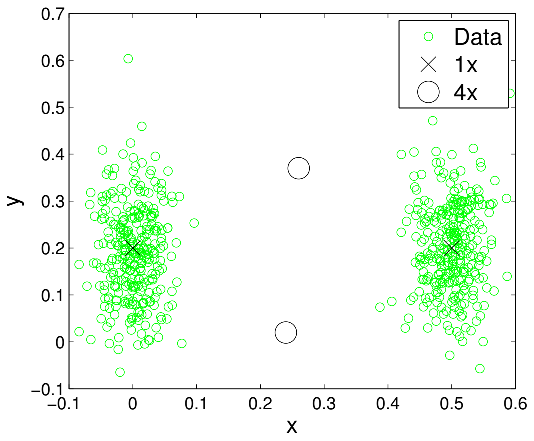


Figure 4.2: Input Gain Error: No error versus $4x$ error.

to variation in the update mechanism, the update rate may vary from one memory cell to another.

An asymmetric error will have some system level impact. During the transient stage of centroids moving towards their clusters, this error will simply cause the learning rate to be modified. During the steady state stage of centroids learning the mean and variance of their clusters, this will cause the mean to be offset within the cluster and the calculated variance to be larger. The expected offset is expected to be proportional to the mismatch between increments and decrements. Assuming the increment is larger than the decrement, if the number of increments and decrements are equal the centroid will move upwards. However, as the centroid moves upwards the number of decrements will increase and cause the centroid to reach equilibrium.

$$\lambda_i Pr(X > \mu) = \lambda_d Pr(X < \mu) \tag{4.6}$$

A demonstration of this effect can be seen in Figure 4.3. As the size of the increments relative to the decrements becomes larger, the centroid becomes more offset from the true center of the data. Depending on the nature of the data being clustered over, the amount of error that is acceptable may vary. It is also important to remember that extremely accurate clustering is not needed, and even with fairly inaccurate clustering,

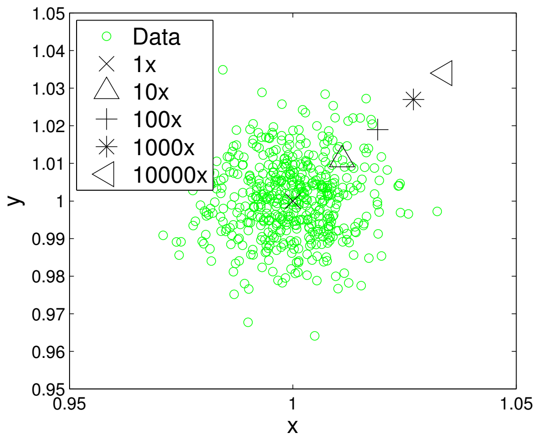


Figure 4.3: Centroid Offset Error: Varying amounts of asymmetry in updates.

meaningful beliefs can be calculated. Variation in the update rate between memory cells will have the effect of some centroids converging towards their equilibrium faster than others. This will have no effect on the equilibrium state of the centroids and no significant effect on the learning transient except in extreme cases.

4.1.3 Memory Adaptation Variation

Each analog memory cell will tend to converge towards the inputs applied to it when it is updated. Input-referred offset or gain error here will cause the memory to converge to a value different than the actual mean of the cluster it is learning. This error can be represented by modifying the error term applied to the observation for the update equation with respect to that applied for the distance measurement.

$$\mu_{x,gain} = \mu_x + \alpha\epsilon_{x,gain}(o - \mu_x) \tag{4.7}$$

$$\mu_{x,bias} = \mu_x + \alpha(\epsilon_{x,bias} + (o - \mu_x)) \tag{4.8}$$

If the memory adaptation variation is modeled as a gain error as in equation 4.7, it will have the effect of increasing or decreasing the learning rate. Much like

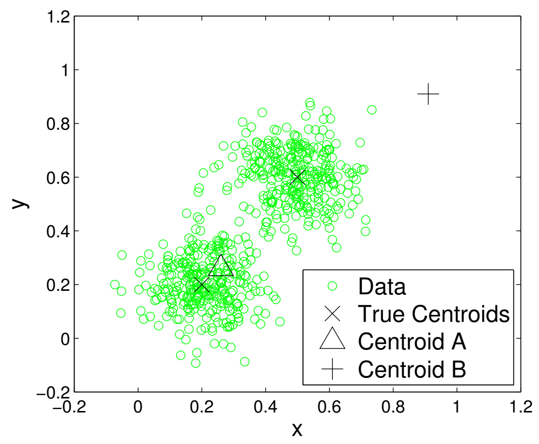


Figure 4.4: Memory Adaptation Bias Error

the update variation error, it will have little to no effect except in extreme cases. However, if this error is modeled as an additive error as in equation 4.8, it will have the effect of shifting the learned centroid by the amount of the error. For small errors, this will have a small effect on the calculation of the belief state. When the error becomes larger, it can cause a centroid to walk far away from the data cluster it is supposed to represent and possibly towards another data cluster. The effect of such a case is demonstrated in Figure 4.4. This figure demonstrates what happens when one centroid has a bias error in the memory adaptation. In this figure, Centroid B has a bias error toward the top right of the plot. As it moves towards the other data cluster, it swaps positions with Centroid A before reaching a steady state position offset from its data cluster by approximately the magnitude of the bias error.

4.1.4 Distance Error

The distance measurement circuits may exhibit gain and/or offset errors. Variation within a given centroid could result in one dimension contributing disproportionately to the overall distance between an observation and an input. Circuitry accepting the individual one-dimensional inputs will contribute to error uncorrelated across dimension within a given centroid. Circuitry operating on the aggregated distance will contribute to an error that affects each dimension identically.

For reasonable error levels, distance error has the effect of giving one dimension more or less importance is the centroid selection process much like the case of the input gain error demonstrated in figure 4.2.

4.1.5 Distance Comparison

There may be input-referred offset in the distance comparison block, typically implemented as either a winner-take-all (WTA) circuit for similarity or a loser-take-all (LTA) for difference. The effect of this can be expected to be similar to a distance error that is identical for all dimensions of a given centroid, but that varies across centroids.

On a system level, distance comparison error will have the effect of causing inaccurate belief state calculations and making a centroid appear artificially further from (or closer to) all inputs in the selection algorithm. If this error is small, it will have no effect since the distance comparison is only used to select the centroid to update. If the error is larger, a centroid could become starved until this distance comparison error comes into equilibrium with the starvation trace. Once this equilibrium is reached, the centroid will still claim fewer input vectors than it should, since its starvation trace cannot remain small enough to claim input vectors without allowing other centroid to claim more input vectors. This effect is demonstrated in Figure 4.5. In this figure, Centroid B has an error causing it to appear artificially more distant from any input. This results in Centroid A to claiming some observations that should belong to Centroid B.

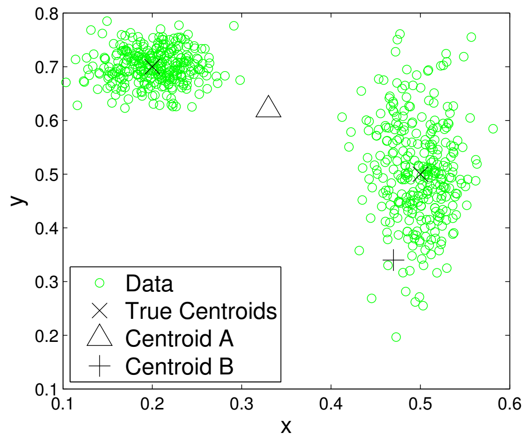


Figure 4.5: Illustration of a distance calculation error within the clustering process.

4.1.6 Additive Noise

Each of the signals is an analog current or voltage and is thus subject to additive noise. The noise is typically Gaussian. It may be spectrally white such as shot noise and thermal noise, or pink concentrated at low frequencies, such as flicker noise.

Noise should have no effect on the calculated centroid means since it is a zero mean process. For clusters with a smaller variance than the noise level, the calculated variance will be similar to the noise level rather than similar to the true variance. If the true variances are less than the noise level for all centroids, then the beliefs will be calculated based on a distance measure that is approximately Euclidean distance than normalized Euclidean distance because of the inaccurate calculated variances. The beliefs will then have less information than they otherwise would, but the beliefs will still be meaningful because of the accurate mean values.

4.1.7 Resulting Generic Clustering Model

Assuming current-mode signaling is the only type used, incorporating the non-ideal effects into ((4.1)-(4.5)) yields the following relationships:

$$d_{i,j} = \frac{(\epsilon^a_{i,j}\epsilon^b_{i,j}o_i - \mu_{i,j} + n^a_{i,j}[t])^2}{\epsilon^c_{i,j}} + n^b_{i,j}[t] \tag{4.9}$$

$$D_j = \sum_{i=0}^{N} \epsilon^d_{i,j} d_{i,j} + n^c_{i,j}[t] \tag{4.10}$$

$$\delta_{i,j} = (\epsilon^a_{i,j}\epsilon^e_{i,j}o_i - \mu_{i,j}) \tag{4.11}$$

$$\mu_{i,j}[t+1] = \begin{cases} \mu_{i,j}[t] + \alpha^+_{i,j}\delta_{i,j} & \text{if } D_j = \min(D_{1:M}) \ \& \ \delta_{i,j} > 0 \\ \mu_{i,j}[t] + \alpha^-_{i,j}\delta_{i,j} & \text{if } D_j = \min(D_{1:M}) \ \& \ \delta_{i,j} < 0 \\ \mu_{i,j}[t] & \text{otherwise.} \end{cases} \tag{4.12}$$

$$\gamma_{i,j} = (\epsilon^f_{i,j} d_{i,j} - \mu_{i,j}) \tag{4.13}$$

$$\sigma^2_{i,j}[t+1] = \begin{cases} \sigma^2_{i,j}[t] + \beta^+_{i,j}\gamma_{i,j} & \text{if } D_j = \min(D_{1:M}) \ \& \ \gamma_{i,j} > 0 \\ \sigma^2_{i,j}[t] + \beta^-_{i,j}\gamma_{i,j} & \text{if } D_j = \min(D_{1:M}) \ \& \ \gamma_{i,j} < 0 \\ \sigma^2_{i,j}[t] & \text{otherwise.} \end{cases} \tag{4.14}$$

$$n_j = \sum_{i=1}^{N} \frac{(\epsilon^a_{i,j}\epsilon^b_{i,j}o_i - \mu_{i,j} + n^a_{i,j}[t])^2}{\sigma^2_{i,j}[t]} + n^b_{i,j}[t] \tag{4.15}$$

4.1.8 Belief State Inaccuracy

While the prior sections focus on the inaccuracies that exist in the core clustering algorithm, this section focuses on an inaccuracy exclusive to the belief state calculation. When the belief state is being calculated, it is necessary to take a vector of normalized euclidean distances, invert them, and normalize the result to sum to 1. The analog circuitry used for this process can result in inaccurate calculations of the belief state vector. During this normalization process, small belief state values are calculated incorrectly. This effect can be modeled by setting any belief state element b_c smaller than a threshold t_n to zero and renormalizing as shown in equation 4.16 to get the resulting belief state element with this inaccuracy b^{err}_c.

$$b_c^{err} = \frac{b_c \mathbb{1}_{b_c > t_n}}{\sum\limits_{c' \in C} b_{c'} \mathbb{1}_{b_{c'} > t_n}} \tag{4.16}$$

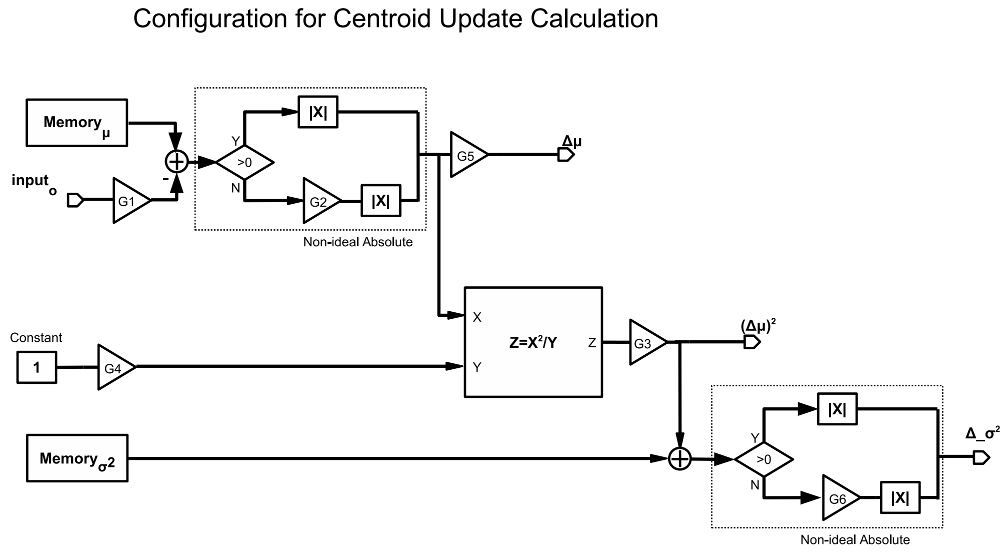
As t_n grows, information from more centroids will be lost. However, a smaller value of t_n will require more power and area on the analog chip. The error model given in Equation 4.16 is guaranteed to be stable as long as t_n is less than $\frac{1}{K}$ where K is the number of centroids in the clustering model. For this reason, only values of t_n less than $\frac{1}{K}$ will be explored in this work since larger values of t_n could introduce instabilities into the system.

4.2 Final Design

Using the results gathered by investigating the generic analog clustering model heretofore, a specific implementation and design was proposed. The proposed design is given in Figure 4.6 on a per dimension per centroid basis. Current-mode signaling is being used for all signals in this circuit. This results in a system where mismatch only arises as gain errors. On a per dimension per centroid basis, six gain errors are inherent to the design.

4.2.1 Errors in Centroid Updates

The first diagram in Figure 4.6 demonstrates the circuit configuration used to update the centroid values. In this section, the effect each of the errors have upon the resulting learned centroids is discussed. The first error, G_1, is an input variation error and will have the effect of altering the importance of each dimension, as was previously discussed in Section 4.1.1. The second error, G_2, will have the effect of an update asymmetry error as discussed in Section 4.1.2 when updating the centroid mean. It will also have an effect on the calculation of the centroid variance. The centroid variance will be miscalculated as shown in Equation 4.17.



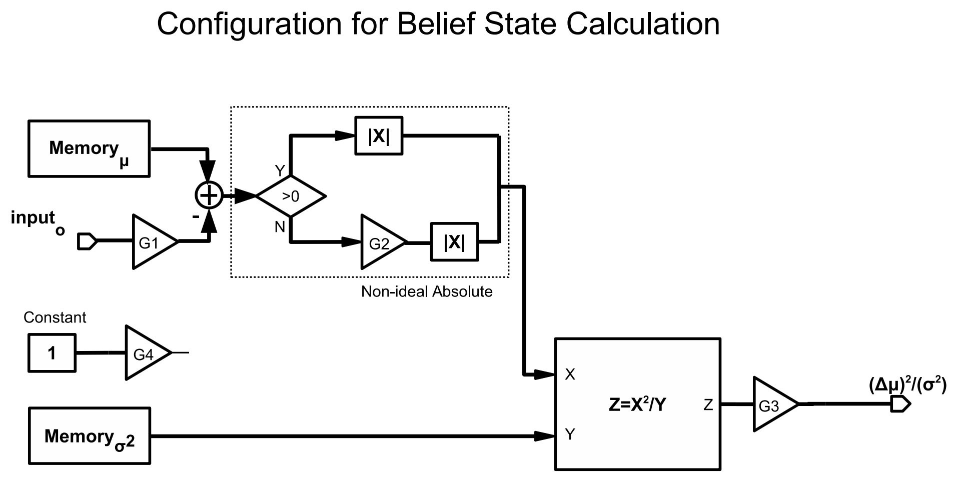


Figure 4.6: This figure gives the design of the analog clustering circuit design. It represents a single dimension of a single centroid. (Top) This diagram shows the configuration of the circuit during the centroid update phase. (Bottom) This diagram shows the configuration of the circuit during the belief state calculation phase.

$$(\sigma^2)_{G_2} \approx \tfrac{1+G_2}{2}\sigma^2 \tag{4.17}$$

The errors G_3 and G_4 will also result in a miscalculation of the centroid variances as shown in Equations 4.18 and 4.19. These errors will also have exhibit the same effects as the distance errors described in Section 4.1.4.

$$(\sigma^2)_{G_3} = G_3\sigma^2 \tag{4.18}$$

$$(\sigma^2)_{G_4} = \tfrac{1}{G_4}\sigma^2 \tag{4.19}$$

The error G_5 has the effect of a memory adaptation variation as presented in Section 4.1.3, and it will have the effect of altering the learning rate of the centroid means on a per dimension per centroid basis. The error G_6 is an update asymmetry in the calculation of the centroid variance. Thus, it will result in a shift in the centroid variances. This shift will depend on the magnitude of G_6 and will be bound by between the maximum and minimum input provided to the non-ideal absolute circuit. The effect this has on the calculated centroid variance is illustrated in the Equations 4.20 - 4.22.

$$\delta = (memory_\mu - input_o)^2 \tag{4.20}$$

$$F_X(y) = Pr(x < y),\ (CDF) \tag{4.21}$$

$$(\sigma^2)_{G_6} \approx F_\delta^{-1}(\tfrac{1}{1+G_6}) \tag{4.22}$$

4.2.2 Errors in Belief State Calculation

Figure 4.6 shows the circuit configuration for both of the tasks at the heart of the DeSTIN node: updating the centroids and calculating the belief state. It demonstrates how the same components are utilized for both of these operations. By reusing the same circuitry for both operations, area on the chip is saved, and the mismatch errors in computation are consistent between the two stages with the exceptions of the error labeled G_4 in Figure 4.6 and to a lesser degree the error labeled G_6. This inconsistency can result in inaccurately calculated centroid variances compared to the distance they are compared to as shown in Equation 4.26. This is in contrast to an error like the one labeled G_3 that exists in both the centroid update process and the belief state calculation process. While G_3 results in a miscalculated centroid variance, its reuse in the belief calculation has results in this error having no impact as long as linearity assumptions on the circuit are still valid. This is demonstrated in Equation 4.25. The following equations demonstrate the effect each error will have upon the normalized euclidean distance calculation for each dimension of each centroid. G_5 is not included because it only has an effect on the learning rate of the centroid means. G_2 can affect a shift in the centroid mean, but this effect is not included below since the focus here is the effect on belief state calculation independent of the centroid means.

$$(\frac{X^2}{Y})_{G_1} = \frac{(G_1X)^2}{G_1^2\sigma^2} = \frac{G_1^2X^2}{G_1^2\sigma^2} = \frac{X^2}{\sigma^2} \tag{4.23}$$

$$(\frac{X^2}{Y})_{G_2} \approx \frac{(\frac{(1+G_2)}{2}X)^2}{(\frac{(1+G_2)}{2})^2\sigma^2} = \frac{(\frac{(1+G_2)}{2})^2X^2}{(\frac{(1+G_2)}{2})^2\sigma^2} = \frac{X^2}{\sigma^2} \tag{4.24}$$

$$(\frac{X^2}{Y})_{G_3} = \frac{X^2}{G_3\sigma^2}G_3 = \frac{X^2}{\sigma^2} \tag{4.25}$$

$$(\frac{X^2}{Y})_{G_4} = \frac{X^2}{\frac{1}{G_4}\sigma^2} = \frac{G_4X^2}{\sigma^2} \tag{4.26}$$

$$\delta = (memory_{\mu} - input_{o})^2 \tag{4.27}$$

$$F_X(y) = Pr(x < y),\ (CDF) \tag{4.28}$$

$$(\frac{X^2}{Y})_{G_6} \approx \frac{X^2}{F_{\delta}^{-1}(\frac{1}{1+G_6})} \tag{4.29}$$

4.3 Effects of Analog Inaccuracies on DeSTIN

Discussion heretofore has focused on the effects analog inaccuracy has upon the learned centroid parameters or upon the calculation of the belief state. In this section, the effect these inaccuracies have upon the learning ability of DeSTIN is discussed. DeSTIN does not rely upon convergence to a specific set of centroid values. It only needs to capture regularities that exist in the data, and map these to belief values that serve as good features for classification. The remainder of this chapter discusses what constitutes a major distortion that will degrade DeSTIN's ability to produce good features.

4.3.1 Effect of Gain Errors

In Section 4.2.2, the effects of the gain errors on the belief state calculation were discussed. Many of these errors had no effect effect on the belief state calculation outside of the effect they have on the learned centroid means, which was discussed in Section 4.2.1. G_1 simply distorts the importance of each dimension in choosing the winning centroid. G_2 causes a shift in the location of the centroid mean. These two errors can result in different centroid locations in the centroid mean locations. G_3 has no effect on the centroid mean and also has no effect on the belief state calculation as shown in Equation 4.25. G_5 simply alters the learning rate for the centroid mean, and thus only has an effect on the rate of convergence. This leaves two gain errors errors that are more likely to have an impact on meaningful belief state calculation. G_6 has an effect such that an inaccurate centroid variance is calculated. This inaccurate centroid variance is then used to calculate the normalized euclidean distance. However, this miscalculated centroid variance is bound between the largest and smallest squared distance between the centroid mean and the input. Thus, while this results in a miscalculated centroid variance, it is still bounded by the actual distances observed. G_4, however, is not exhibit this type of behavior. G_4 can result in centroid variances that are artificially small (or large) and are only bound by the size of the error. If this error grows too large, the calculated normalized distances are not based on the variances that are actually occurring. When this happens, the centroid variances don't capture which dimensions are highly varying and which are not. Thus, two points that should produce different belief states since they are very different in a low-variance dimension produce more similar belief states. This results in lower-variance centroids being produced in the next layer than should be produced. This is reflected in experimental results presented in Chapter 5.

4.3.2 Effect of Belief Normalization Error.

The belief normalization error presented in Section 4.1.8 could also have an effect on the classification performance of DeSTIN. As the threshold is increased, the information gained from from the centroids further away from the current point will be lost. Without this error, the belief state formed will be based on the relationship between the input and all K centroids. With this error, the belief state will not contain information on the relationship between the input and the furthest centroids. As the threshold t_n grows larger, the belief state will lose information about the relationship between the input and the centroids furthest from it. However, the most important belief state elements are the one based on the closest centroids which means DeSTIN should be relatively insensitive to this error. The error is explored experimentally in Chapter 5.

4.3.3 Effect of Additive Noise

Noise in the system is undoubtedly going to cause information in the belief state to be lost. However, as state previously, there is not a *correct* belief state, and what an acceptable change in the belief state is must be determined. All information is expected to be lost if the variance of the noise is larger than the variance of the clusters that would be formed without noise. At this point, any difference between two inputs is going to be a result of noise and not a result of a meaningful difference between the points. In other words, as the variance of the noise grows larger than the variance of the clusters, the signal to noise ratio (SNR) approaches $0dB$.

$$SNR = \frac{\sigma^2_{centroid}}{\sigma^2_{noise}} \tag{4.30}$$

However, information will be lost before the noise grows that large. It is more useful to know how much noise can be withstood without losing significant information. As the noise level rises compared to the variance of the centroids, it will begin to have an effect on the belief states produced. This will result in the next clustering layer operating in an altered space. In recurrent clustering, this increased noise will lead to the more regularities existing in the intrinsic dimensions comparatively to the extrensic dimensions. As a result, all points will produce more similar belief states and the mean distance between centroids (MDBC) in the next layer will decrease. This results in increased overlap in the space represented by the centroids as shown in Equation 4.31 where A_c gives the area that is less than one standard deviation from the mean of centroid c. In Convolutional DeSTIN, this recurrence does not exist and the added noise will simply result in more wildly varying belief states and the mean distance between centroids in the next layer will increase. As the MDBC increases, the closest centroid will begin to dominate the belief state resulting in a belief state the holds less information. As the MDBC decreases, there will increased overlap and centroids are less able to characterize meaningful differences in the input. Experimental results obtained in Chapter 5 demonstrate this.

$$Overlap = \sum_{c \in C} \sum_{c' \neq c \in C} A_c \cup A'_c \tag{4.31}$$

4.3.4 Effects of Additive Noise on Depth

DeSTIN is a multi-layered architecture which makes it important to understand how noise interacts with higher layer features. The higher layers operate on the belief states of the lower layers. Thus, as the belief states from the lower layers degrade with increasing noise, the regularities captured at the higher layers will be less meaningful. This results in belief states from the higher layers being less meaningful for classification since clustering assumptions begin to fail (i.e. points in the same cluster share meaning). This means that as the belief states from the first layer begin to significantly degrade as the amount of noise increases, the belief states from the upper layers will not be able to capture any meaningful information from the layers below. Thus, as the MDBC for the second layer beliefs begin to change as described in Section 4.3.3, the information contained in the upper layer beliefs will be minimal, and classification performance will be near guessing. Thus, it is expected that performance using beliefs from only the upper layers will approach guessing at the same point classification using beliefs from all layers begins to significantly decrease. This is reflected in results from Chapter 5. This is an important effect to understand, since future work will focus on using only belief states from the upper layers unlike current work that performs classification based on belief states collected from all layers of DeSTIN.

Chapter 5

Experimental Results

In this Chapter experimental results for the methods introduced in Chapter 3 and and of the effect of the analog inaccuracies presented in Chapter 4 are presented.

5.1 Standard Datasets

In this section the standard datasets used in the research will be introduced. The reasons for their use will be explained, and their characteristics will be detailed.

5.1.1 MNIST Dataset

One of the benchmarks presented here is the MNIST handwritten digit dataset (Lecun and Cortes, 2009). While current literature in the field of deep learning focuses largely on the CIFAR-10 dataset, results on MNIST are also frequently presented. This dataset is also used here as a tool to compare the results of the improved DeSTIN architecture to that of previous implementations. This dataset is used in such a manner as to create ”synthetic” temporal data by scanning a viewing window over the images such that the images are viewed as a sequence of smaller images. Thus, the recurrent clustering mechanism must characterize the series of small images it is presented with. This dataset has been used extensively in the literature and contains

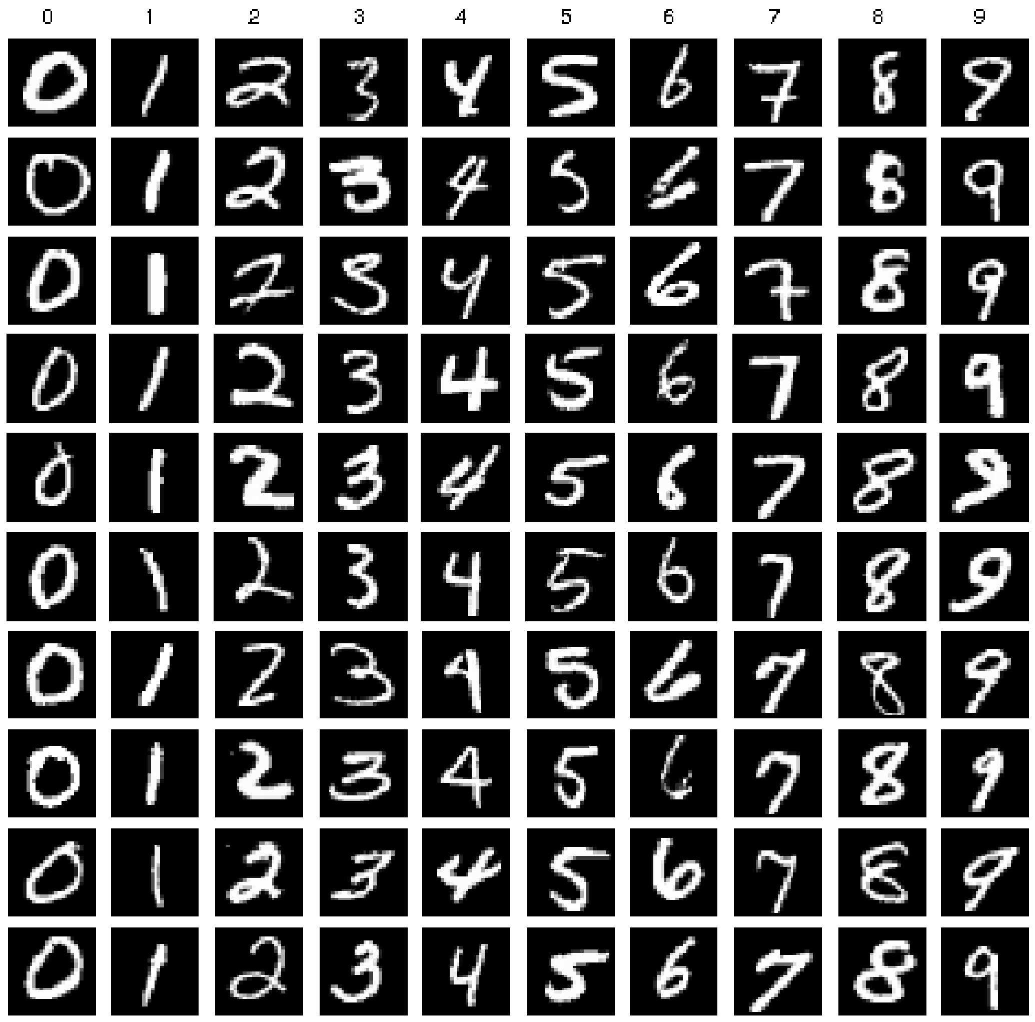


Figure 5.1: MNIST Image Examples. There are 10 classes of images, and each image is a 28x28 gray-scale image.

60, 000 training images of digits 0-9 and 10, 000 testing images. The images are 28×28 pixels in size, roughly centered, and are gray-scale. It is important to note that although the images are gray-scale, most values are near saturated to black or white. A random sampling of images from each class is provided in Figure 5.1.

Unless stated otherwise, DeSTIN is configured as described here. The scanning sequence in Figure 5.2 is used with 3 movements being saved. There are 3 layers of DeSTIN nodes with 1, 4, and 16 nodes in each layer respectively. Each node is

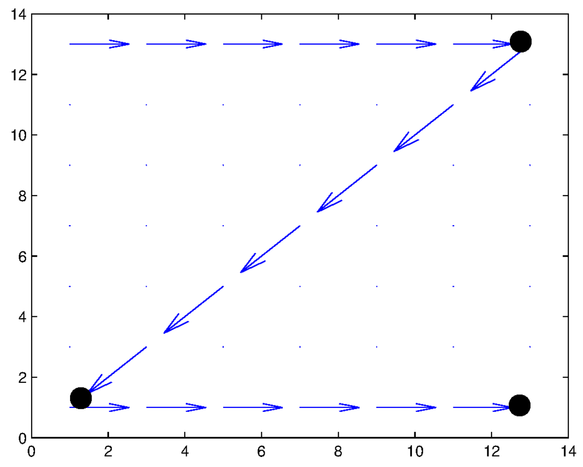


Figure 5.2: MNIST "Z" Movement Sequence: The black dots indicate movements from which belief states are saved.

resourced with 25 centroids. Each node in the bottom layer is provided with a 4×4 pixel patch from the image. Recurrent clustering is used.

5.1.2 PEMS-SF Dataset

The proposed method was also tested on the PEMS-SF database (Cuturi, 2011). This dataset is used because it contains natural temporal features. Unlike the MNIST dataset where temporal features are generated by scanning a viewing window across an image, this dataset contains sensor data that varies over time. This dataset gives the relative occupancy rate of many lanes of traffic on the San Francisco area freeways collected over 15 months. Public holidays and two days that were missing data were not included in the dataset. The data from 963 sensors was collected over 440 days every 10 minutes and the task is to classify each day as the correct day of the week (e.g. Monday). The dataset consists of 267 training samples and 173 testing samples.

For use in this work, the data from each sensor was normalized to have zero mean and unit variance. Only 576 (24×24) of the 963 sensors were used in our tests to accommodate the 24×24 "viewing window" of the bottom layer and only 33 time-steps, every $4th$ starting from the beginning, were provided to the system. Now, instead of using the next movement over an image, the input layer is provided with data from the next time-step. Then, a feature vector was produced from the belief state of every node for every $11th$ time-step. This resulted in a feature vector made up of belief states that characterized each third of the day. The DeSTIN hierachy is configured in the same manner as for the MNIST dataset, with the exception that each node has a 6×6 dimensional input.

5.1.3 CIFAR-10 Dataset

The CIFAR-10 dataset (Krizhevsky and Hinton, 2009) is the most widely used benchmark for deep learning. It contains $50,000$ training examples and $10,000$ test examples. Each example is a 32×32 color image. There are 10 classes: airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks. These images are a subset of the 80 million tiny images dataset (Torralba et al., 2008). A random sampling of images from each class is provided provided in Figure 5.3.

Unless otherwise stated, when using the CIFAR-10 dataset DeSTIN is resources similarly to when using the MNIST dataset. The major difference is that Convolutional DeSTIN is used instead of the temporal sampling scheme. With this dataset, the normalization and ZCA whitening scheme outlined in Section 2.3 is performed on each 4×4 pixel path provided to the bottom layer nodes. As previously stated, this preprocessing step is performed in order to account for the strong correlation between neighboring pixels.

5.2 Demonstrative Results

In this section demonstrative results verifying the methods and statements presented in Chapter 3 are presented.

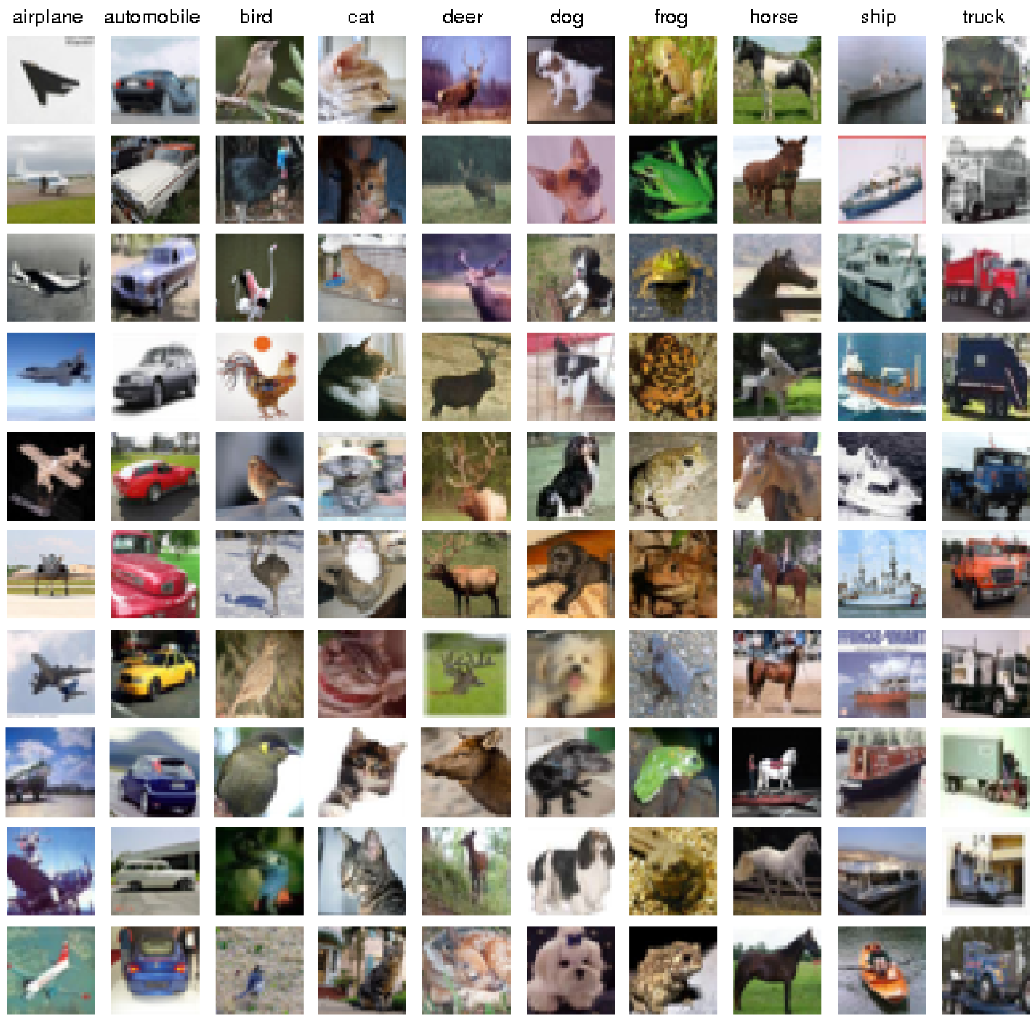


Figure 5.3: CIFAR-10 Image Examples. There are 10 classes of images, and each image is a 32×32 RGB image.

5.2.1 Recurrent Clustering Temporal Representation Abilities

In this section, experiments that demonstrate the ability of recurrent clustering to capture temporal information are presented. First, its capabilities will be explored in depth in order to demonstrate its contribution in extracting temporal information from data even when the time scales of the important information is large. Consequently, the performance of the algorithm within a fully-hierarchical structure, as applied to a standard benchmark, will be presented.

A key attribute expected of recurrent clustering is recognition of patterns across time. As means of demonstrating this capability, the proposed recurrent clustering algorithm is applied to time-series prediction tasks. The first task explored is a frequency doubler where the objective is for system to take as input a sampled sinusoid signal with a period of N and to produce belief states that can be used as features to a simple feed-forward neural network whose output should be a sinusoid with half the period. This problem requires that the belief state captures information that at least spans the current and previous inputs.

Figure 5.4 depicts the results of the frequency doubler test case. The algorithm was run with 24 centroids and the feed-forward neural net is resourced with 32 hidden neurons. As can be seen, the incremental algorithm was easily able to create features capturing temporal dependencies even for fairly slow, small changes taking place in the input, as reflected by larger periods. When the input signal period became too large, prediction error began to grow as a result of the small differences between samples which are challenging to represent using limited centroids. However, the resulting prediction remains consistently better than a random guess.

The second experiment was targeted at the algorithm's ability to capture temporal attributes, particularly in the context of detecting a binary sequence of interest within a general stream of binary inputs. The goal was to demonstrate the property of latching on to long-term temporal regularities. The length of the sequence of interest

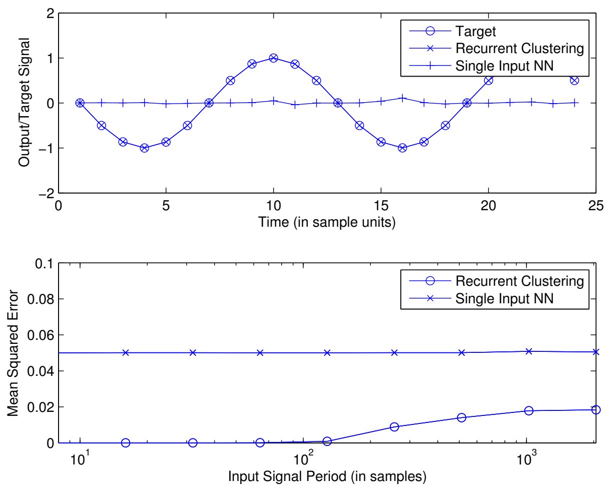


Figure 5.4: Frequency Doubler results: (Top) The target vs. output plot is given for recurrent clustering and for a case where only the current value of the input is provided to the clustering algorithm. The period of the original signal is 24 sample units. (Bottom) The ability of the algorithm to capture information in long period sine waves is evaluated.

was varied in order to observe the impact of long sequences on the accuracy of the algorithm. The sequence of interest was a randomly chosen binary sequence of specific length. To further increase the challenge at hand, the overall input sequence was generated by randomly selecting (with probability 0.5) either the sequence of interest or the sequence of interest with the first binary element inverted. The belief states for the sequences of interest and the sequence with only the first bit altered were provided to a feed-forward neural network for the purpose of classifying each sequence. If the belief states accurately learn to represent regularities in the sequences presented, the classifier should be able to achieve a classification rate of 100%. A purely random selection (i.e. guessing) is represented by a classification rate of 50%.

Classification results for the sequence detection task are presented in Figure 5.5 for varying sequence lengths and number of centroids. The classification rate observed decays exponentially with the length of the sequence, which is anticipated as a result of the unsupervised nature of the algorithm. Since there is no supervision that guides the algorithm to best identify a sequence of any specific length, the beliefs will always

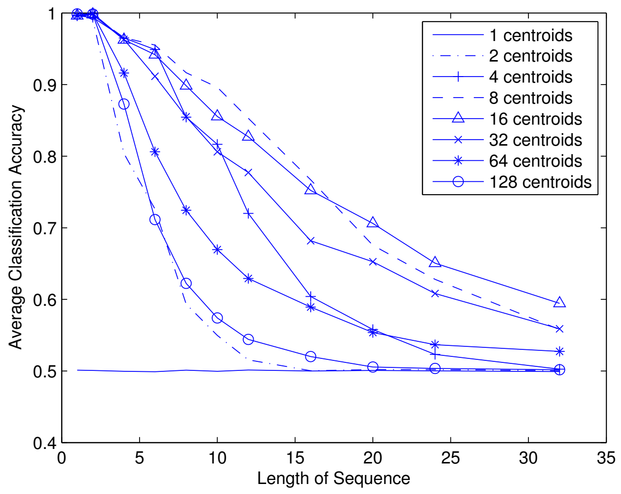


Figure 5.5: Sequence Detection Results: This plot illustrates the average classification accuracy as a function of the length of the sequence of interest. The recurrent clustering algorithm is resourced with a varying number of centroids.

hold more information about more recent observations. The results indicate that there is an optimal range for the number of centroids used where the algorithm performs best. In the case of too few centroids, the belief state may not capture long time spans, while if there are too many centroids, the belief state may represent features in the data that are not relevant for identifying the sequence of interest. However, the algorithm exhibits weak sensitivity to the number of clusters, which is a desired property.

5.2.2 Supervised Clustering

In Section 3.3, the importance of selecting a proper label weight was discussed. It was explained that a label weight that was too large would cause overlap in centroids since centroids would be formed that contained no differences in the non-label dimensions. This would manifest itself in increased centroid variances. Figures 5.6 - 5.8 illustrate the classification performance and average centroid variance for each of the standard benchmarks.

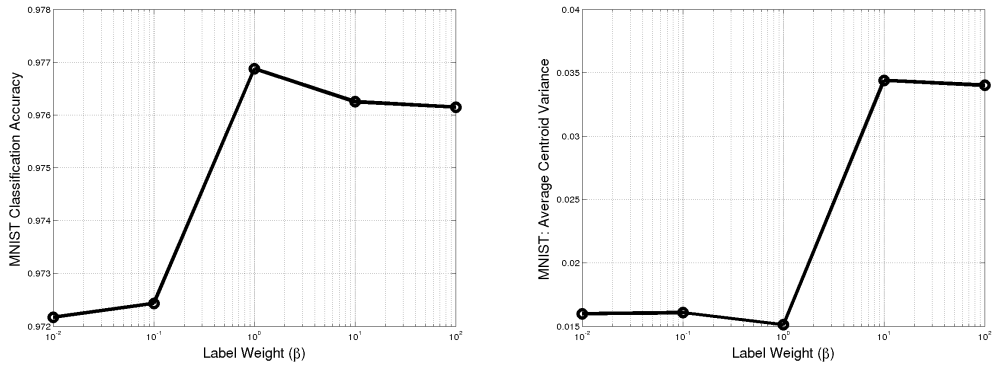


Figure 5.6: MNIST results using supervised clustering. The classification accuracy (left) and the average centroid variance (right) are plotted against the label weight β.

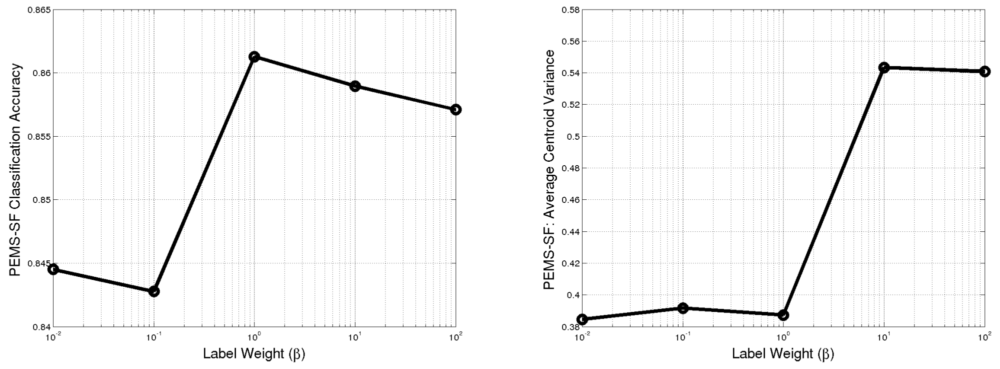


Figure 5.7: PEMS-SF results using supervised clustering. The classification accuracy (left) and the average centroid variance (right) are plotted against the label weight β.

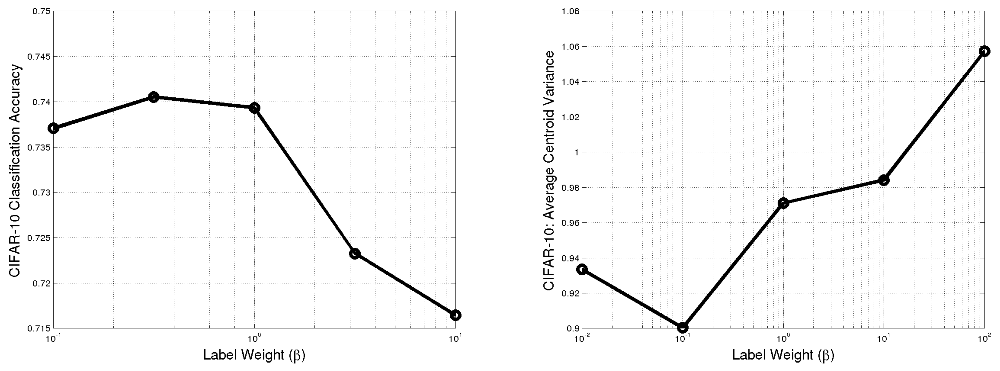


Figure 5.8: CIFAR-10 results using supervised clustering. The classification accuracy (left) and the average centroid variance (right) are plotted against the label weight β. 75 centroids per node were used to obtain these results.

These figures demonstrate that the increase in cluster variance is indicative of declining classification performance. This increase in cluster variance is the result of the label weight, β, placing too much importance on the creating centroids that represent a single class. This results in centroids that are near duplicates of each other in the non-label dimensions and thus fewer unique centroids must represent the same space that was previously represented with more unique centroids. Thus, the average centroid variance increases. The PEMS-SF dataset appears to benefit the most from this supervised clustering technique with a increase in classification performance of 1.5% over using unsupervised clustering, while the other datasets observe a smaller increase in performance. The MNIST dataset saw a 0.5% increase in performance, while the CIFAR-10 dataset saw only an 0.25% increase in performance.

5.3 Maximum Performance on Standard Benchmarks

In this section, the best results obtained on the standard benchmarks are presented. These results are then compared to other state of the art methods and previous results using DeSTIN.

5.3.1 MNIST Results

The $60,000$ training images were elastically deformed(Simard et al., 2003) in order to form an additional $120,000$ training images.

The DeSTIN hierarchy employed consisted of 3 layers with 4×4 nodes in the bottom layer, 2×2 nodes in the middle layer, and 1 node at the top layer. The movement sequence used is shown in Figure 5.9. Each of the nodes in the bottom layer received a different 4×4 pixel patch of the input image, which results in the bottom layer viewing a 16×16 window during each movement. Nodes in every layer hosted a different number of centroids with the bottom, middle, and top layers having

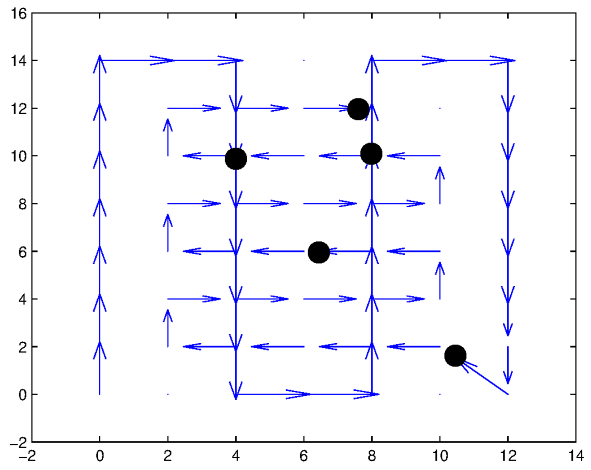


Figure 5.9: Large movement sequence used for better performance on MNIST.

32, 24, and 32 centroids, respectively. For training purposes, a random sampling of $15,000$ of the training set images was used. Only 15,000 images are used since the clustering algorithm only needs to be able to accurately calculate the mean and variance of each centroid. Thus, as long as there are enough samples to accurately characterize the regularities in the data, no benefit will be gained by training on additional samples. Next, all $180,000$ training images and $10,000$ testing images were provided to the DeSTIN network in order to generate feature vectors for each image. A feature vector for each image consisted of the belief state of every node in the hierarchy sampled at every $12th$ movement. These feature vectors were then provided to an supervised classifier in order to obtain classification results.

The supervised classifier consisted of an ensemble of 11 feed-forward neural networks trained with negative correlation learning . Each network hosted two hidden layers with 128 and 64 hidden neurons respectively and was trained to predict the posterior probability distribution over the classes. The cross-entropy error function was used in conjunction with a softmax output activation function, which ensured that the network outputs were within the range [0,1] and summed to one. All $180,000$ training feature vectors (both elastic and non-elastic) were used in training, and inputs

Table 5.1: Comparison of Results on MNIST

Method	Classification Accuracy
DeSTIN (previous) (Karnowski, 2012)	98.55%
DeSTIN (this work)	98.71%
Single Deep Neural Network (no width normalization) (Schmidhuber, 2012)	99.53%
Multi-Column Deep Neural Network (committee of 35 networks each trained on data with different width normalization) (Schmidhuber, 2012)	99.77%

Table 5.2: Comparison of Results on PEMS-SF

Method	Classification Accuracy
AR-Kernel (Cuturi and Doucet, 2011)	75%
AR-Kernel using k (Cuturi and Doucet, 2011)	81%
BOV Kernel (Cuturi and Doucet, 2011)	82%
GA Kernel (Cuturi and Doucet, 2011)	79%
SS Kernel (Cuturi and Doucet, 2011)	81%
Kernels in Reservoir Space (Chen et al., 2013)	86%
DeSTIN	86%

to the networks were scaled to the range [-1, 1]. Using this experimental setup, a classification accuracy of 98.71% was achieved which is comparable to previous work involving the first-generation DeSTIN architecture which involved an additional layer and more complex computations. These results are compared to results obtained for this benchmark achieved with other state of the art methods (Kégl and Busa-Fekete, 2009; Salakhutdinov and Hinton, 2007; Simard et al., 2003; Schmidhuber, 2012) in Table 5.1.

5.3.2 PEMS-SF Results

In this section, results on the PEMS-SF traffic sensor dataset are presented. The DeSTIN hierarchy used was the same as before, except it had 50, 30, and 20 centroids in the nodes in the three layers respectively. A classification accuracy of 86% was achieved, which is comparable to other state of the art methods (Cuturi, 2011; Cuturi and Doucet, 2011; Chen et al., 2013) as can be seen in Table 5.2. This method performed on par with state of the art results on this dataset and has demonstrated an ability to perform well on two very different types of datasets without any complex preprocessing or any significant changes in the method to handle these differences. This ability to discover structure in many different types of data without significantly altering the dataset to fit the method being used is important to the proliferation of DML methods into new problem domains. In particular, it demonstrates DeSTIN's ability to capture temporal features that exist in data.

5.3.3 CIFAR-10 Results

For this test, DeSTIN was configured as before with the exception of the number of centroids per node. 100 centroids per node were used. A classification accuracy of 76% was reached on the test set. This is compared to other methods in Table 5.3. Horizontal reflections of the training set are used to augment the training set, but no scaling or shifts are performed on the data.

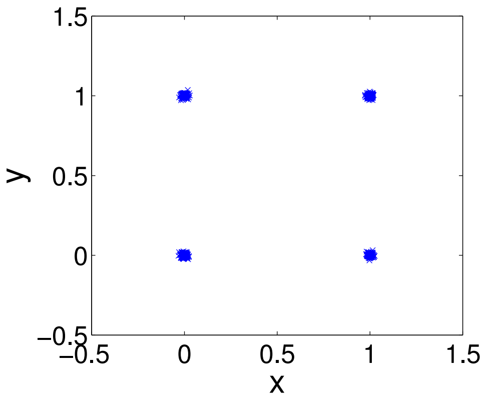
While DeSTIN performs comparable to earlier convolutional neural network results (Ngiam et al., 2010), it is still behind current state of the art results (Wan et al., 2013). A discussion of some possible causes for this difference is discussed in Chapter 6 along with some paths forward to improving DeSTIN's performance. It is important to note DeSTIN's much faster training time. In a single threaded CPU implementation, DeSTIN's centroids can be trained 25x faster than a CNN model can be trained. This is largely a factor of DeSTIN's convergence speed.

Table 5.3: Comparison of Results on CIFAR-10

Method	Classification Accuracy	Single Thread CPU Runtime	GPU Runtime
DeSTIN (Non-Convolutional; No Translated Images)	49%	n/a	n/a
Tiled Convolutional Neural Networks (Ngiam et al., 2010)	73%	n/a	n/a
DeSTIN (Convolutional; No Translated Images)	76%	1.6 hours †	n/a
Convolutional Deep Belief Network (Krizhevsky, 2010)	79%	n/a	81 hours
CNN + DropConnect (Single Network; No Translated Images) (Wan et al., 2013)	81%	41 hours ††	25 minutes
CNN + DropConnect (12 Networks; Translated Images) (Wan et al., 2013)	91%	n/a	n/a

†Time to train DeSTIN centroids

††Estimated time based on GPU run time and a comparison between GPU implementation and single threaded CPU implementation of CNNs (Scherer et al., 2010).



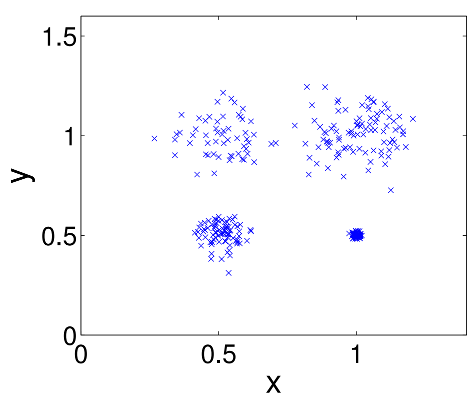


Figure 5.10: Clean (left) and noisy (right) synthetic clustering data used for evaluation of analog computation inaccuracies

5.4 System-Level Impacts: Modeling Inaccuracies

This section focuses on experimental results when simulating the effects of analog circuitry as presented in Chapter 4. Results from both the generic analog clustering model and the specific implementation used are presented.

5.4.1 Generic Clustering Model Results

In this section the impact of error sources on the belief state calculation using the when using the generic analog clustering model presented in Section 4.1. First, a metric for algorithmic performance must be defined. Since the common method for using DeSTIN in pattern recognition tasks is to extract the beliefs as features, performance is defined as the mean absolute error (MAE) between what the ideal belief values should be and those calculated considering errors in the system. It is important to note that this means performance is not directly tied to the numerical accuracy of the calculated centroid means and variances, which are being calculated in a space altered by the error sources. It is only tied to the ability to produce the correct belief state. In the remainder of this section, the effects of the analog error sources on a synthetic dataset in order to demonstrate the effect on calculated belief states when the true centroids are known are explored.

To demonstrate the effect of the various errors and mismatches, a simple clustering problem is considered. The data shown in Figure 5.10 is clustered using a single DeSTIN node with varying levels of error and noise. The noise is always additive Gaussian noise, while Gain errors and additive errors are implemented according to Equations (5.1) and (5.2), respectively, where r_x is the difference between the maximum and minimum values x can take. Noise is added to select signals in the system in the same manner as additive error. The use of currents to represent variables generally leads to gain errors, while voltage-based signals lead to offset errors. The system was modeled as full current-mode signaling and full voltage-mode signaling to explore the different effects. The impact of gain and offset errors on a system using mixed-mode signaling can be expected to fall between these two cases.

$$x' = x\mathscr{N}(1, \sigma) \tag{5.1}$$

$$x' = x + \mathscr{N}(0, \sigma) r_x \tag{5.2}$$

The resulting error in the belief is calculated as the mean absolute error between the ideal belief vector and that obtained using analog computation. Figures 5.11 and 5.12 illustrate the effect of the errors discussed in this section on a single node's belief values. As can be observed, none of the errors introduce any notable degradation below a standard deviation value of 10^{-3}.

When modeling all errors as gain errors, additive noise in the system has a much larger impact than even the rest of the errors combined. Thus, it is demonstrated that the inconsistency caused by noise is much more harmful than the consistent gain and bias errors. The most destructive gain errors are the distance comparison, distance, and memory adaptation variation errors. The update asymmetry, input, and update variation errors have much less impact.

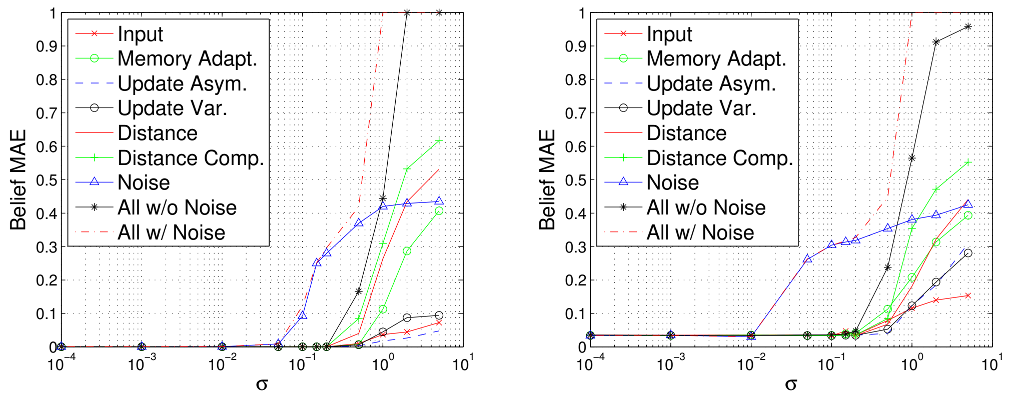


Figure 5.11: Accuracy vs. level of error (σ): Gain Errors on Clean Dataset (left) and Noisy Dataset (right): This figure illustrates that the update and input errors have the lowest impact on performance, while noise has the most significant impact.

When all errors are modeled as bias errors, additive noise has much the same effect as the distance comparison, distance, and memory adaptation variation errors. The update asymmetry, input, and update variation errors still have much less impact.

In conclusion, the results presented here provide a comparison between the mismatch errors that are expected along with noise in the system. This allowed work to focus on the errors expected to have the largest impact when the final analog circuit was designed. In the next section, the final implementation proposed by the analog design team is evaluated.

5.4.2 Evaluation of Final Design

Results in the previous section were used to design a specific analog implementation of the clustering circuit as described in Section 4.2. In this section, the sensitivity of that specific implementation to error and noise is examined. The results in this section focus on classification accuracy on the standard benchmarks. This is a more useful tool for evaluation, since *correct* clustering is not necessary for DeSTIN to produce meaningful beliefs. DeSTIN need not converge to a specific set of centroids. Figures 5.13 - 5.15 contain the classification results on the three standard benchmarks with each of the individual gain errors, all gain errors, noise, and all gain errors

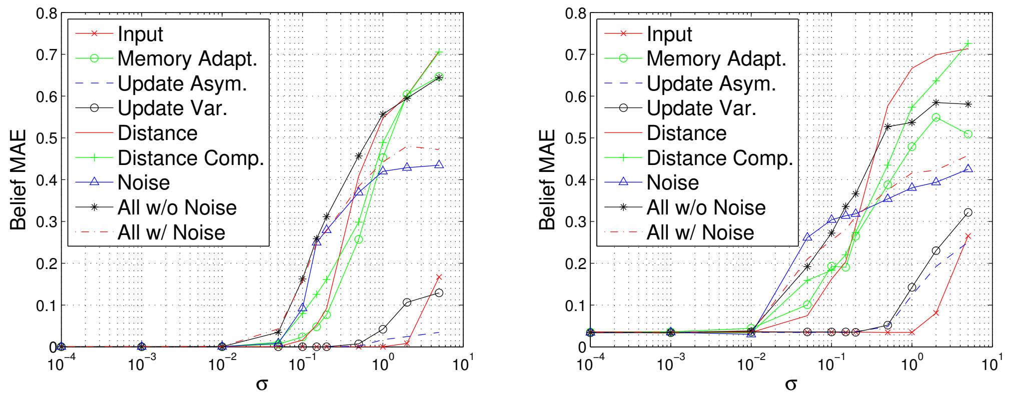


Figure 5.12: Accuracy vs. level of error (σ): Additive Errors on Clean Dataset (left) and Noisy Dataset (right): the update and input errors have the lowest impact, while the remaining error components have an impact similar to that of the additive noise of the same level.

with noise. The classification results are plotted against the standard deviation of the errors/noise in nano-amperes in order to relate the inaccuracies to a physical value. This is calculated by converting the dynamic range of the signals to the typical dynamic range of the signals on an analog chip.

The results from these classification tests on the three standard datasets allow some important conclusions to be drawn. A significant amount of error and noise can be introduced to the DeSTIN architecture without having a destructive effect upon performance. It is particularly noteworthy that noise is the most harmful source of inaccuracy by a significant margin. This is intuitively reasonable, as noise represents a dynamic distortion to which the learning system cannot adapt. In contrast the other error sources distort the signals in a static way, leaving relationships in the underlying data intact.

Effect of Gain Errors

As seen in Figures 5.13 - 5.15, the only gain error with a significant impact on classification accuracy is G_4. As discussed in Section 4.3.1, this is a result of this error only existing in the centroid training phase and not in the belief state generation

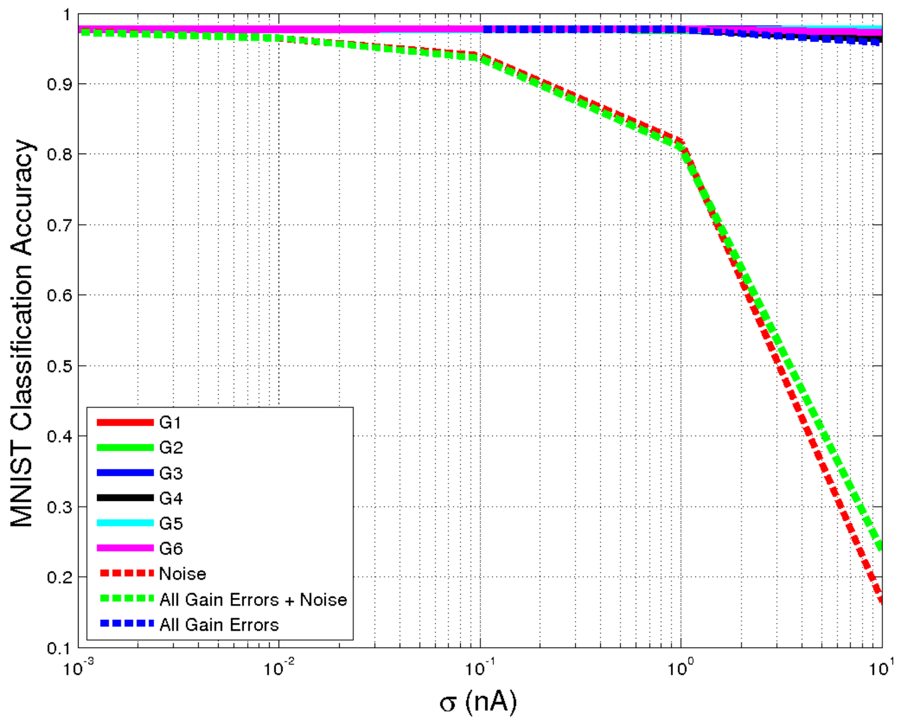


Figure 5.13: Analog MNIST Classification Results

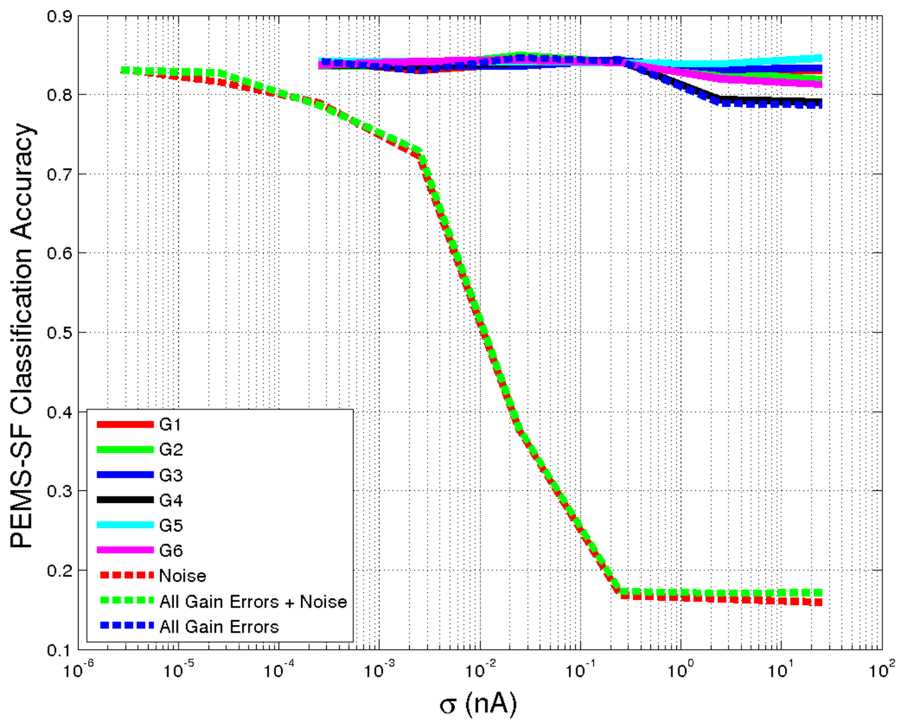


Figure 5.14: Analog PEMS-SF Classification Results

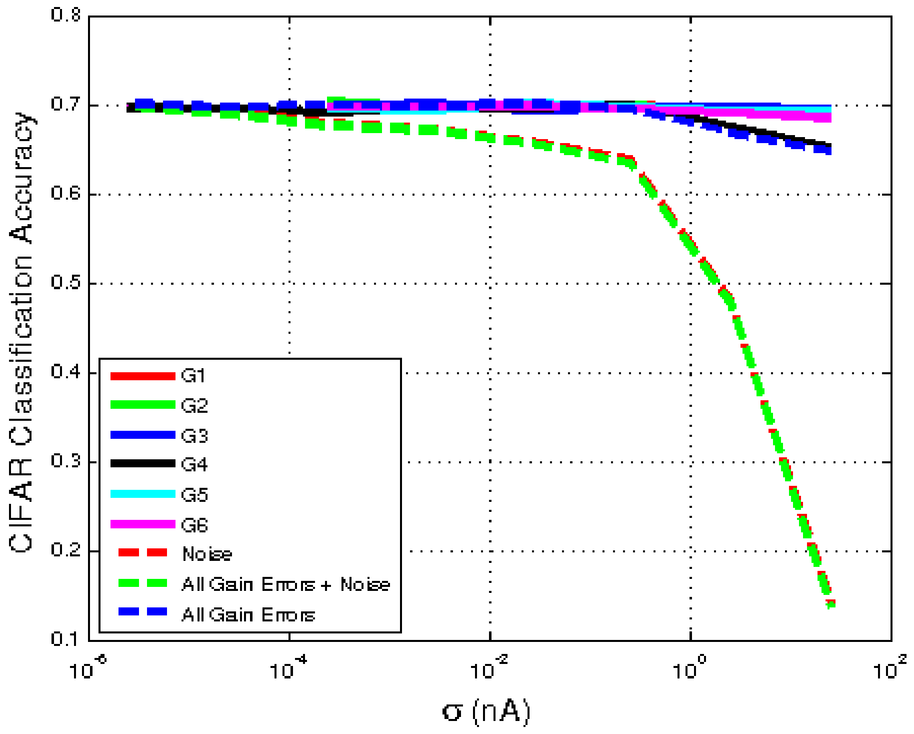


Figure 5.15: Analog CIFAR-10 Classification Results

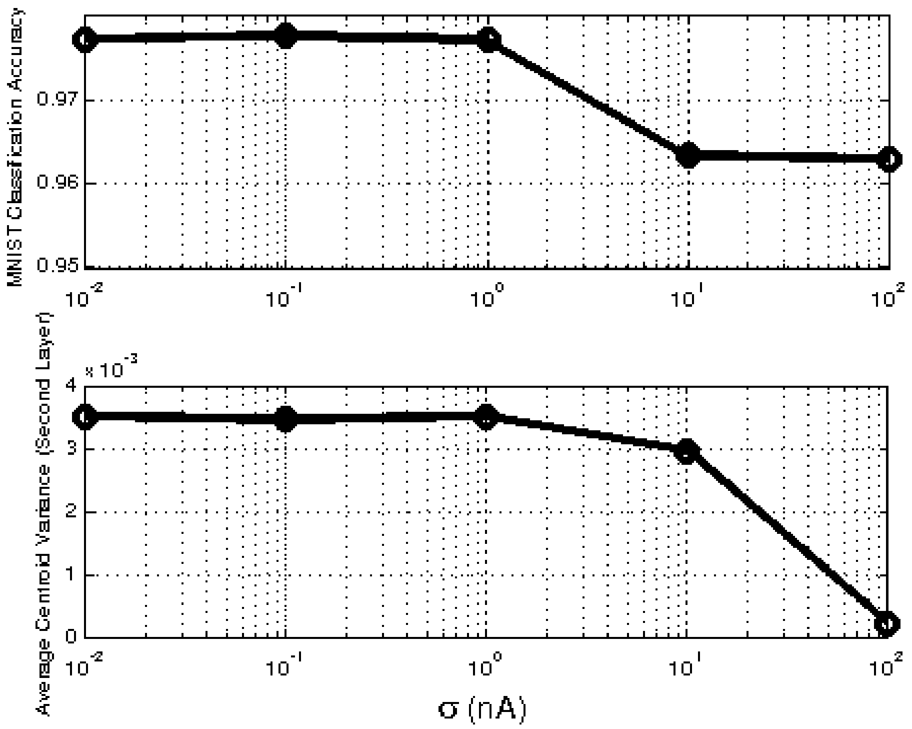


Figure 5.16: G_4 Error and MNIST: Classification results compared to Second Layer Centroid Variance

phase. Figures 5.16 - 5.18 demonstrate the low-variance centroids produced in upper layers as a result of the inaccurately calculated belief states from lower layers as this error grows too large.

Effect of Belief Normalization Error

In this section the belief normalization error previously discussed in Section 4.3.2 is explored experimentally. The threshold t_n was varied in the range $[1.0 \times 10^{-6}, \frac{1}{K}]$ where the the model for this error is guaranteed to be stable. In this range there was no degradation in classification accuracy. Figure 5.19 demonstrates what percentage of belief state elements are non-zero for varying values of t_n.

Since there is no degradation in results in this range, Figure 5.19 demonstrates that the belief states from only 30% of centroids need to be calculated as non-zero.

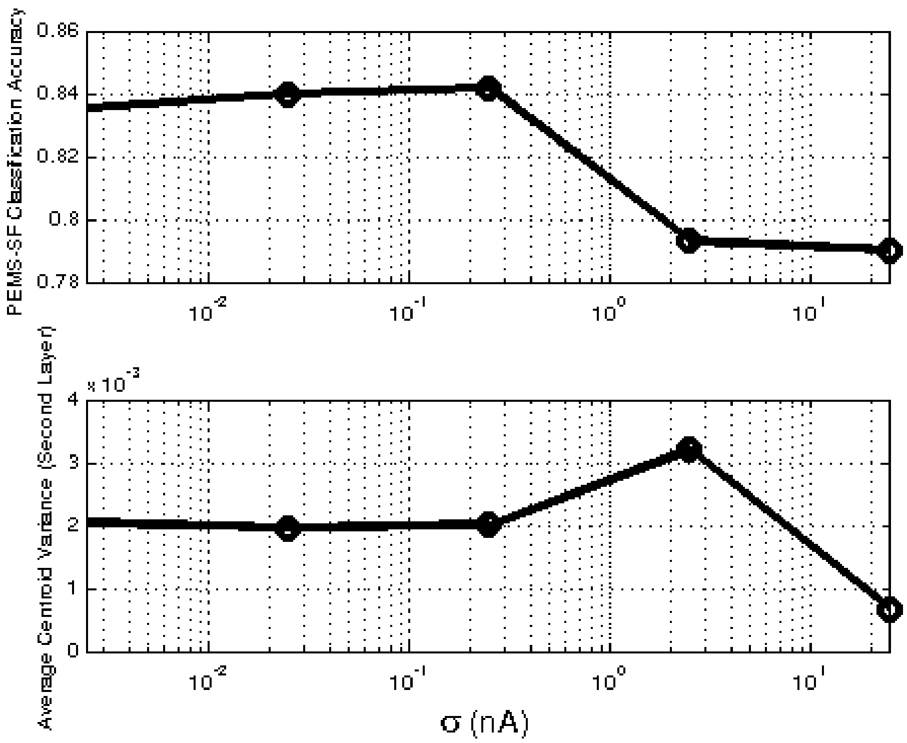


Figure 5.17: G_4 Error and PEMS-SF: Classification results compared to Second Layer Centroid Variance

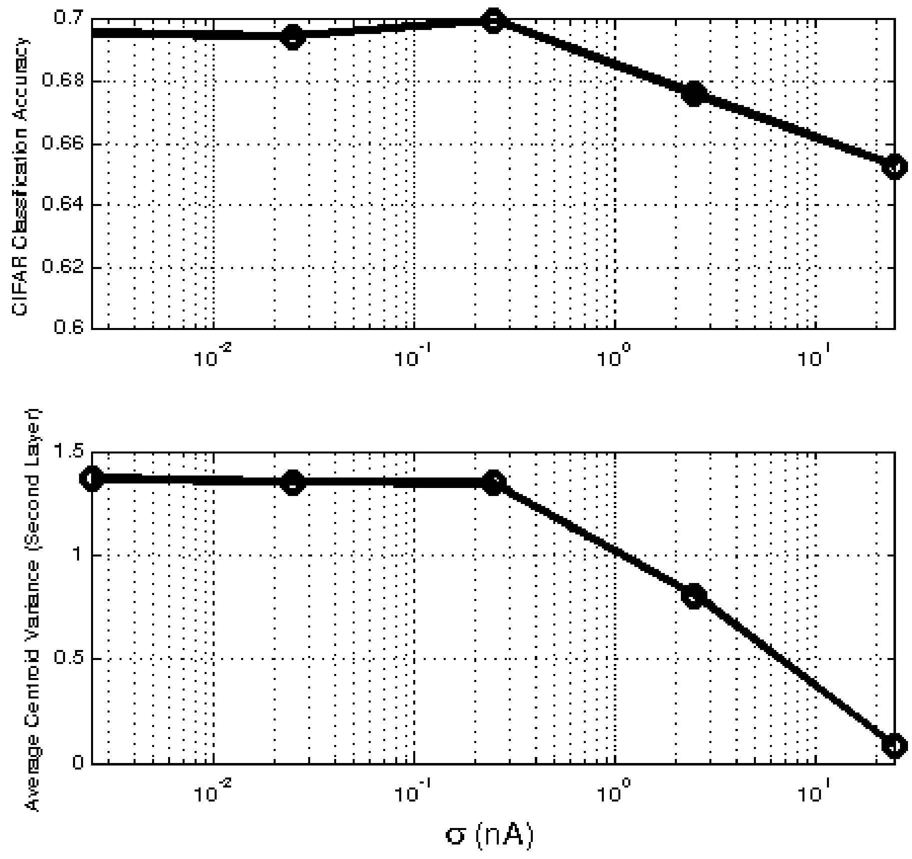


Figure 5.18: G_4 Error and CIFAR-10: Classification results compared to Second Layer Centroid Variance

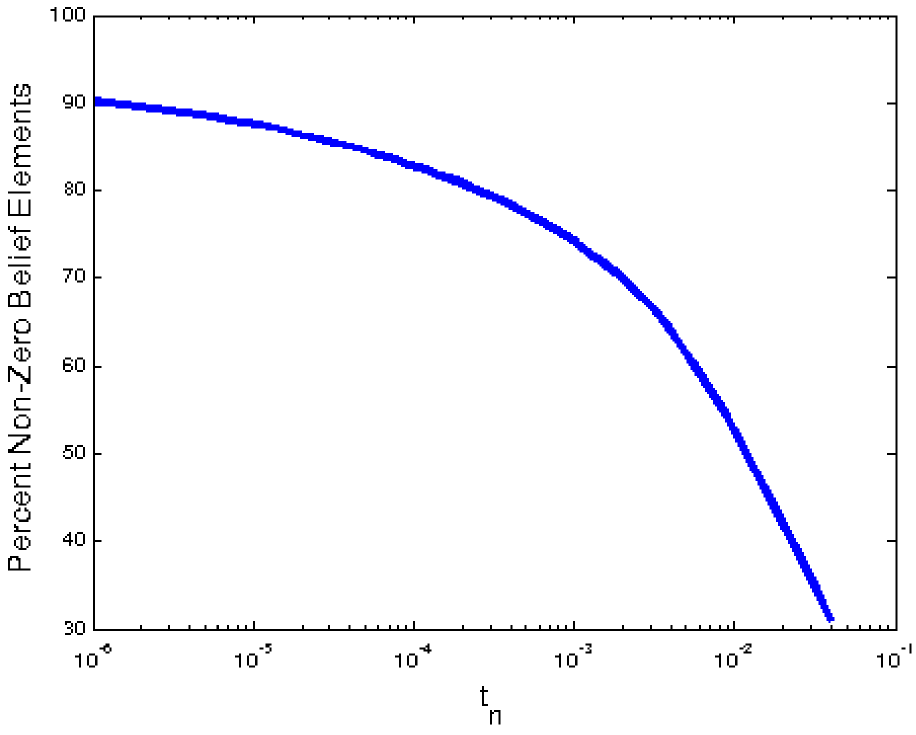


Figure 5.19: Percent non-zero belief elements versus t_n

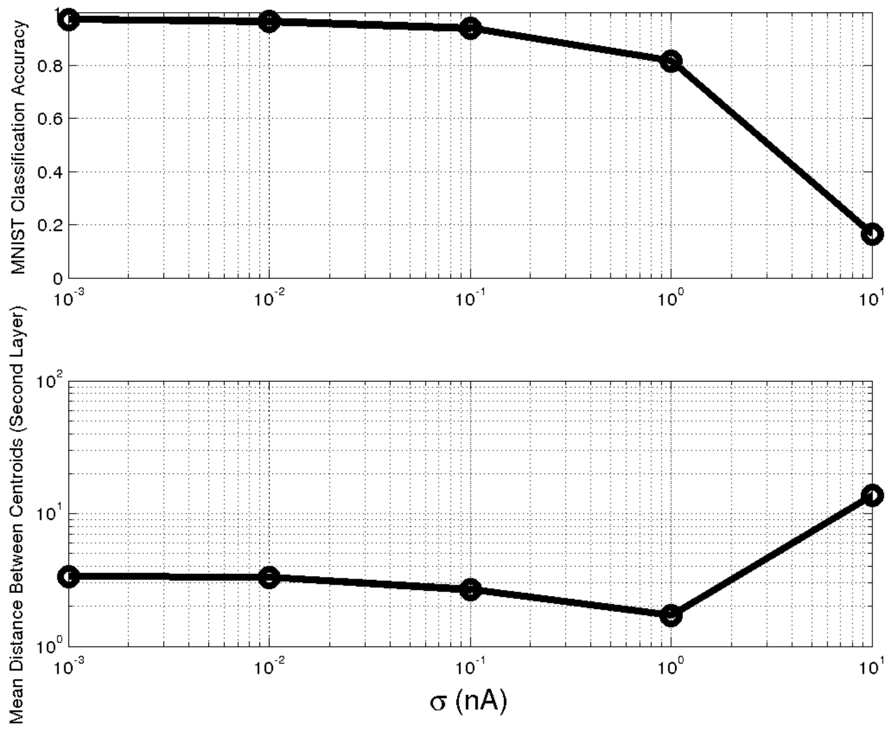


Figure 5.20: Noise and MNIST: Classification results compared to Second Layer MDBC

This means that in a typical DeSTIN configuration consisting of 25 centroids per node, the belief state elements for only 8 centroids need to be correctly calculated as non-zero.

Effect of Additive Noise

As seen in Figures 5.13 - 5.15, noise is has the most significant impact upon classification accuracy on all three datasets. This dynamic distortion of the signals cannot be learned by the system in any meaningful way. Figures 5.20 - 5.22 demonstrate the changes in the mean distance between centroids in the upper layers along with the decrease in classification performance as the amount of noise increases as described in Section 4.3.3. The ”elbow” in the classification accuracy curves occurs when the MDBC experience $2x$ change from the MDBC with no noise.

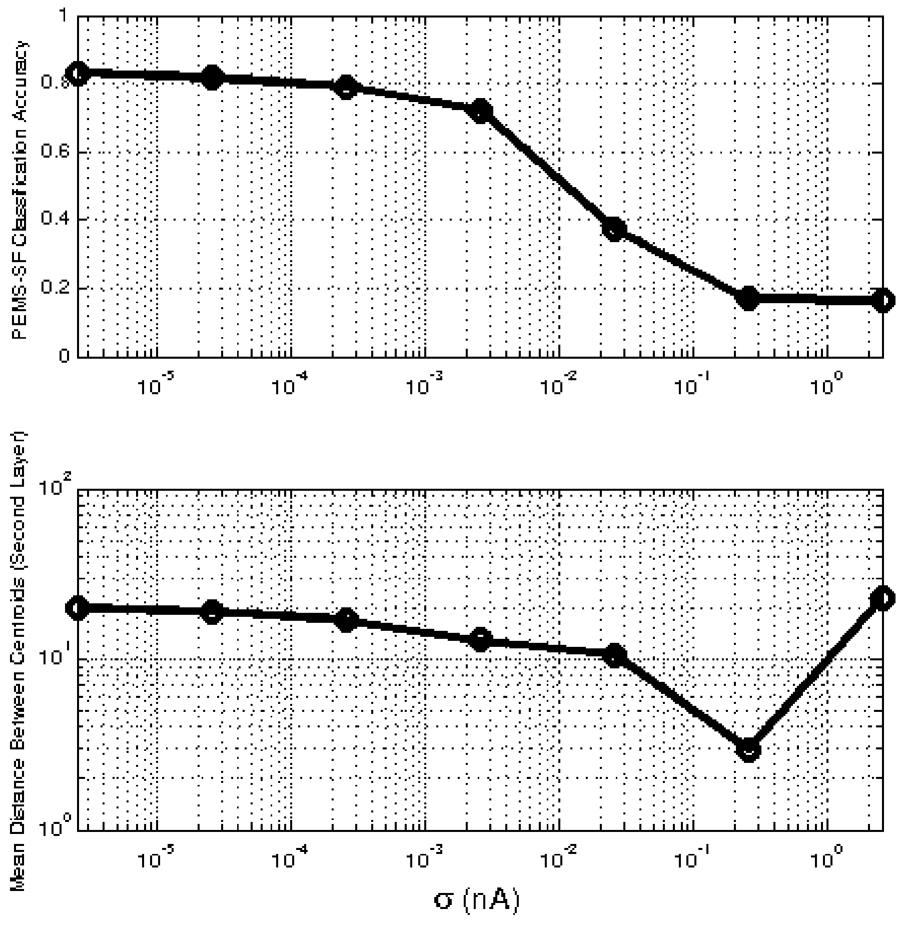


Figure 5.21: Noise and PEMS-SF: Classification results compared to Second Layer MDBC

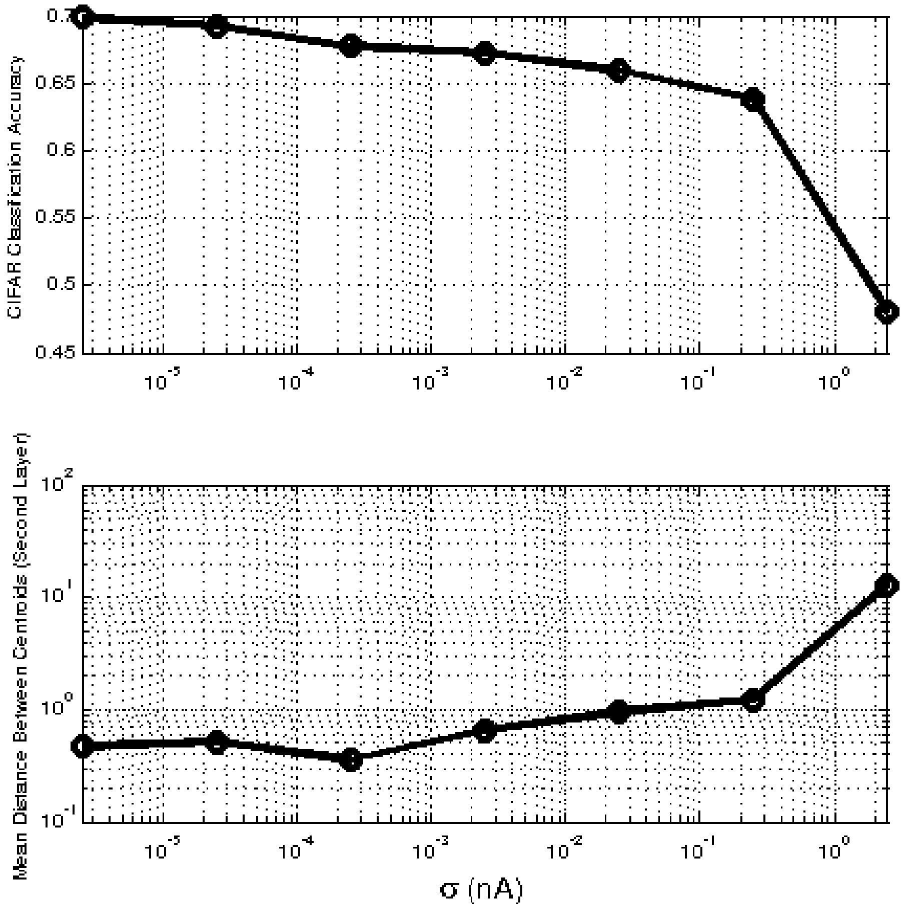


Figure 5.22: Noise and CIFAR-10: Classification results compared to Second Layer MDBC

Table 5.4: Noise Degradation to Guessing

Dataset	$\sigma_{centroids}$ (First Layer)	σ_{noise} (Guessing First Observed)
MNIST	1.0×10^0 nA	1.0×10^1 nA
PEMS-SF	8.0×10^{-2} nA	2.5×10^{-1} nA
CIFAR-10	2.5×10^0 nA	2.5×10^1 nA

Additionally, Table 5.4 demonstrates when classification degrades to guessing. As predicted in Section 4.3.3, this occurs once the standard deviation of the noise is larger than the average standard deviation of the learned centroids when no noise is present. While the results presented in Figures 5.20 - 5.22 are undoubtedly more important, these results explain the upper limit of noise that can be tolerated while still retaining any information at all.

Effect of Additive Noise on Depth

It is important to understand the effect of noise upon the features generated in higher layers of DeSTIN. Figures 5.23 - 5.25 compare classification results using just the upper layer belief states and using belief states from all layers. As explained in Section 4.3.4, classification based only on the belief states from the upper layers will decrease to guessing before classification based on belief states from all layers. Guessing is reached at approximately the point that MDBC centroids significantly changes as seen in Figures 5.20 - 5.22.

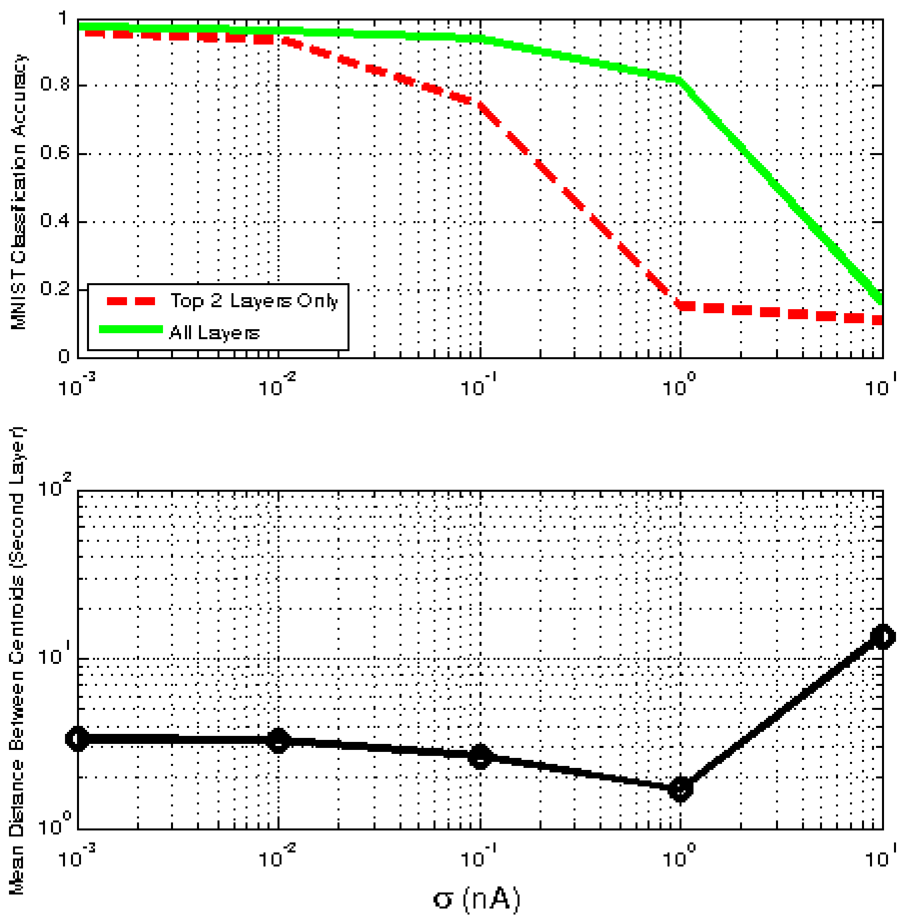


Figure 5.23: Depth,Noise, and MNIST: Classification results using only upper layer beliefs compared to using beliefs from all layers.

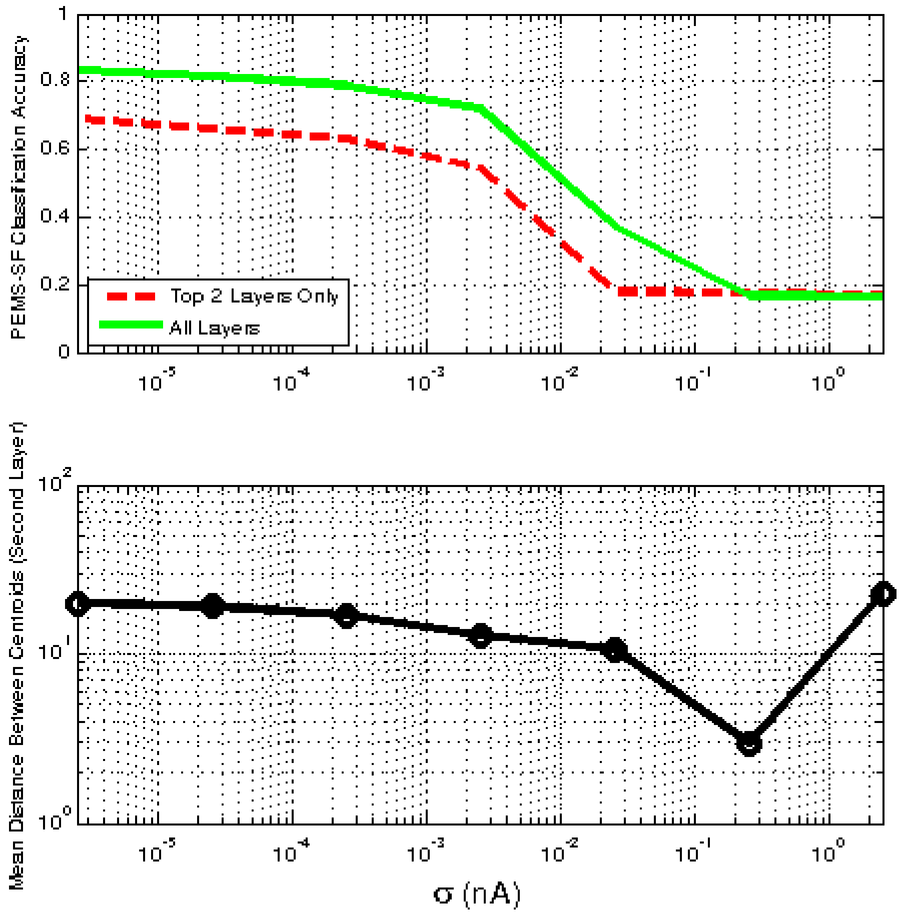


Figure 5.24: Depth,Noise, and PEMS-SF: Classification results using only upper layer beliefs compared to using beliefs from all layers.

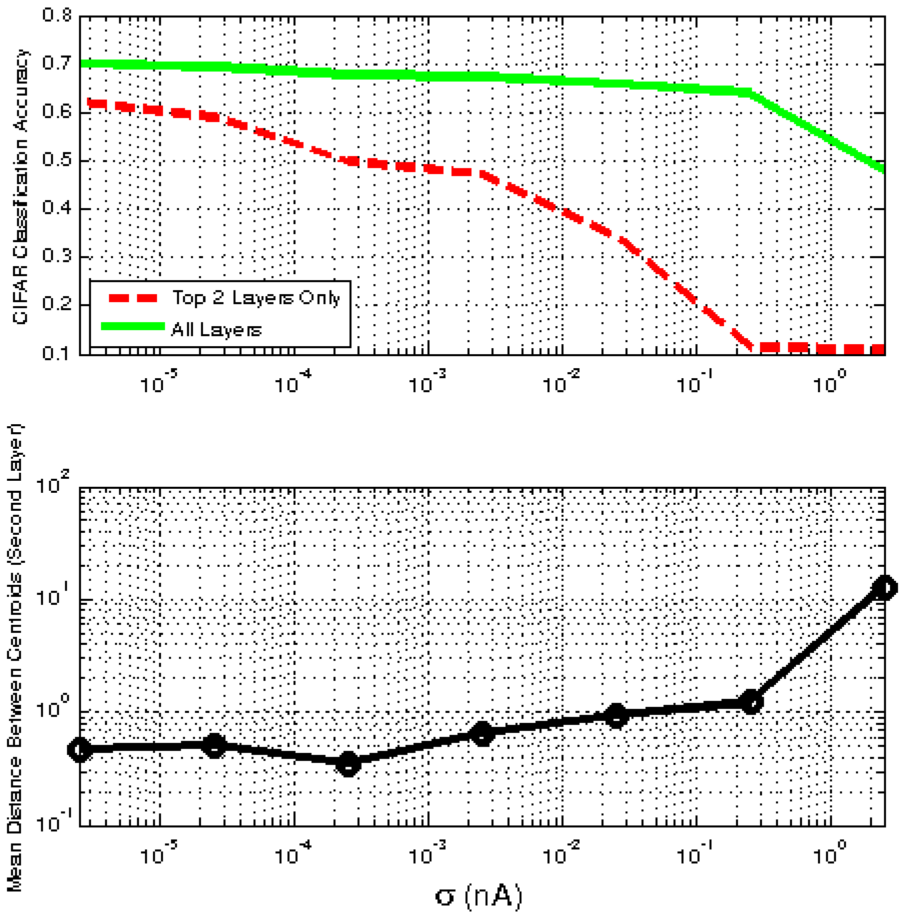


Figure 5.25: Depth,Noise, and CIFAR-10: Classification results using only upper layer beliefs compared to using beliefs from all layers.

Chapter 6

Discussion and Future work

6.1 Discussion

In this work, a new temporal feedback scheme for DeSTIN was introduced. This new method allows DeSTIN to be more easily mapped to alternative hardware by condensing more functionality into the incremental clustering algorithm that is the basic function of each node. A method for using class labels to form centroids more relevant to the classification task being performed was introduced. The label weight, β, given to the class label in the clustering process had previously not been explored in related work. This work demonstrated the effect β has upon classification and provided a method for determining the proper value of this parameter based purely on clustering statistics.

This work presented a method for achieving meaningful results on natural images (CIFAR-10) by borrowing the ideas of convolution and sub-sampling in order to gain shift invariance. Providing meaningful results on the CIFAR-10 dataset was very important since it is the most widely used benchmark for deep learning methods, and MNIST's use as benchmark in deep learning has declined since human-level performance has been achieved.

Finally, this work explored the feasibility of a DeSTIN implementation in analog circuitry and explored the effects of analog inaccuracies on DeSTIN's ability to produce good features for classification. This work was used to guide the design of an analog chip that contains a small-scale DeSTIN hierarchy for proof of concept purposes. The large computation requirements of deep learning methods make implementations in custom architecture very attractive, and analog hardware provides an opportunity to drastically decrease the power consumption of deep learning which has heavily relied on GPUs in recent years.

6.2 Future Work

DeSTIN's classification performance on the CIFAR-10 is still short of state of the art results. DeSTIN relies upon clustering to generate good features for a classifier. Using supervised clustering, the centroids can be forced to correlate with the classes in order to create features that better separate the data for classification. However, CNNs are not simply using the labels to make features correlate with classes. They use the actual classification error to adjust the weights of the network. Future work should include a scheme to leverage classification error to modulate the learned centroids in order to improve classification. This could take the form of a back-propagation method to fine-tune the centroids, or some other form of modulating the clustering process based on the clustering error.

DeSTIN has demonstrated an ability to perform classification on datasets with spatial features (CIFAR-10) and datasets with temporal features (PEMS-SF). The next frontier will be video datasets. Applying DeSTIN to video would likely require convolution across a frame and recurrence of beliefs between frames. Some work has been done applying convolutional neural networks to video classification tasks (e.g. football game vs. track meet), but results using multiple frames versus a single frame on this task were mixed (Karpathy et al., 2014). This is likely because a single frame is all that is required to recognize most of these scenes. Video datasets that contain more important temporal features should be explored. The videos used should not be able to be classified based on a single frame and the order of events should be important. One interesting application to explore would be sign language gesture recognition. Gesture recognition tasks contain strong spatial and temporal information, and it would be very impactful to accomplish for sign language what has been accomplished in deep learning already for spoken language.

Bibliography

Abdelazeem, S. (2009). A novel domain-specific feature extraction scheme for arabic handwritten digits recognition. In Wani, M. A., Kantardzic, M. M., Palade, V., Kurgan, L. A., and Qi, Y., editors, *International Conference on Machine Learning and Applications, ICMLA 2009, Miami Beach, Florida, USA, December 13-15, 2009*, pages 247–252. IEEE Computer Society. 19

Arel, I., Rose, D., and Coop, R. (2009). Destin: A scalable deep learning architecture with application to high-dimensional robust pattern recognition. In *Proc. AAAI Workshop on Biologically Inspired Cognitive Architectures*, pages 1150–1157. 25

Arel, I., Rose, D. C., and Karnowski, T. P. (2010). Deep machine learning–a new frontier in artificial intelligence research. *IEEE Computational Intelligence Magazine*, 5(4):13–18. 21, 24

Bell, A. J. and Sejnowski, T. J. (1996). Edges are the" independent components" of natural scenes. In *NIPS*, pages 831–837. 21

Bellman, R. (1957). *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition. 2

Bengio, Y. (2009). Learning deep architectures for ai. *Foundations and Trends® in Machine Learning*, 2(1):1–127. 2, 20, 21, 22

Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828. 2

Bengio, Y. and LeCun, Y. (2007). Scaling learning algorithms towards ai. *Large-Scale Kernel Machines*, 34. 2, 20

Bialek, W., Rieke, F., Ruyter, D., and Warland, D. (1991). Reading a neural code. *Science*, 252(5014):1854–1857. 5

Bourlard, H. and Kamp, Y. (1988). Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics*, 59(4-5):291–294. 20

Brin, S. and Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. *Comput. Netw. ISDN Syst.*, 30(1-7):107–117. 8

Bryll, R., Gutierrez-Osuna, R., and Quek, F. (2003). Attribute bagging: improving accuracy of classifier ensembles by using random feature subsets. *Pattern recognition*, 36(6):1291–1302. 46, 47, 113

Campbell, M., Hoane Jr, A. J., and Hsu, F.-h. (2002). Deep blue. *Artificial intelligence*, 134(1):57–83. 8

Chapelle, O., Schölkopf, B., Zien, A., et al. (2006). *Semi-supervised learning*, volume 2. MIT press Cambridge. 18

Chen, C.-L., Chen, W.-C., and Chang, F.-Y. (1993). Hybrid learning algorithm for gaussian potential function networks. *Control Theory and Applications, IEE Proceedings D*, 140(6):442–448. 41

Chen, H., Tang, F., Tino, P., and Yao, X. (2013). Model-based kernel for efficient time series analysis. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, pages 392–400, New York, NY, USA. ACM. 77, 78

Cuturi, M. (2011). Fast global alignment kernels. In Getoor, L. and Scheffer, T., editors, *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pages 929–936, New York, NY, USA. ACM. 68, 78

Cuturi, M. and Doucet, A. (2011). Autoregressive Kernels For Time Series. *ArXiv e-prints*. 77, 78

Day, W. and Edelsbrunner, H. (1983). *Efficient algorithms for agglomerative hierarchical clustering methods*. Forschungsberichte: Institute für Informationsverarbeitung. Inst., TU, Computerges. 11

de Ruyter van Steveninck, R. R., Lewen, G. D., Strong, S. P., Koberle, R., and Bialek, W. (1997). Reproducibility and variability in neural spike trains. *Science*, 275:1805–1809. 5

Duda, R. O., Hart, P. E., and Stork, D. G. (2000). *Pattern Classification (2nd Edition)*. Wiley-Interscience. 2, 9, 11, 13, 14, 16, 18

Erhan, D., Manzagol, P.-A., Bengio, Y., Bengio, S., and Vincent, P. (2009). The difficulty of training deep architectures and the effect of unsupervised pre-training. pages 153–160. 4

Farabet, C., LeCun, Y., Kavukcuoglu, K., Culurciello, E., Martini, B., Akselrod, P., and Talay, S. (2011). Large-scale fpga-based convolutional networks. *Machine Learning on Very Large Data Sets*. 4

Farabet, C., Martini, B., Akselrod, P., Talay, S., LeCun, Y., and Culurciello, E. (2010). Hardware accelerated convolutional neural networks for synthetic vision systems. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pages 257–260. IEEE. 4

Farabet, C., Poulet, C., and LeCun, Y. (2009). An fpga-based stream processor for embedded real-time vision with convolutional networks. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, pages 878–885. IEEE. 4

Gerbrands, J. J. (1981). On the relationships between svd, klt and pca. *Pattern recognition*, 14(1):375–381. 19

Gonzalez, R. C. and Woods, R. E. (1992). *Digital Image Processing*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition. 18

Hamel, P. and Eck, D. (2010). Learning features from music audio with deep belief networks. In *Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR)*, pages 339–344. 3

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800. 24

Hinton, G. E., Deng, L., Yu, D., Dahl, G. E., Mohamed, A., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. N., and Kingsbury, B. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Process. Mag.*, 29(6):82–97. 3

Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554. 21, 23

Ho, T. K. (1998). Nearest neighbors in random subspaces. In *Lecture Notes in Computer Science: Advances in Pattern Recognition*, pages 640–648. Springer. 46, 47

Hof, R. D. (2013). Deep learning. *MIT Technology Review*, 116(3). 1

Hyvrinen, A., Hurri, J., and Hoyer, P. (2009). Principal components and whitening. In *Natural Image Statistics*, volume 39 of *Computational Imaging and Vision*, pages 93–130. Springer London. 21

Jiang, X. (2009). Feature extraction for image recognition and computer vision. In *Computer Science and Information Technology, 2009. ICCSIT 2009. 2nd IEEE International Conference on*, pages 1–15. 19

Karnowski, T., Arel, I., and Rose, D. (2010). Deep spatiotemporal feature learning with application to image classification. In *Machine Learning and Applications (ICMLA), 2010 Ninth International Conference on*, pages 883 –888. 30

Karnowski, T. P. (2012). *Deep Machine Learning with Spatio-Temporal Inference.* PhD thesis, The University of Tennessee, Knoxville, Tennessee. 27, 30, 77

Karnowski, T. P., Arel, I., and Young, S. (2011). Modeling temporal dynamics with function approximation in deep spatio-temporal inference network. In *BICA*, pages 174–179. 32

Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., and Fei-Fei, L. (2014). Large-scale video classication with convolutional neural networks. In *Proceedings of International Computer Vision and Pattern Recognition (CVPR 2014).* 99

Kégl, B. and Busa-Fekete, R. (2009). Boosting products of base classifiers. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 497–504, New York, NY, USA. ACM. 77

Krizhevsky, A. (2010). Convolutional deep belief networks on cifar-10. 79

Krizhevsky, A. and Hinton, G. (2009). Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto.* 23, 45, 69

Le, Q. V., MarcAurelio Ranzato, R. M., Matthieu Devin, K. C., and Greg Corrado, J. D. (2012). Andrew ng (2012b).building high-level features using large scale unsupervised learning. In *International Conference in Machine Learning (cit. on p. 131).* 1

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324. 22

Lecun, Y. and Cortes, C. (2009). The MNIST database of handwritten digits. 23, 45, 66

Lee, H., Grosse, R., Ranganath, R., and Ng, A. Y. (2009a). Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 609–616, New York, NY, USA. ACM. 21

Lee, H., Grosse, R., Ranganath, R., and Ng, A. Y. (2009b). Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 609–616. ACM. 24

Lee, H., Grosse, R., Ranganath, R., and Ng, A. Y. (2011). Unsupervised learning of hierarchical representations with convolutional deep belief networks. *Commun. ACM*, 54(10):95–103. 23

Lockett, A. J. and Miikkulainen, R. (2009). Temporal convolution machines for sequence learning. Technical Report AI-09-04, Department of Computer Sciences, the University of Texas at Austin. 24

Lu, J., Young, S., Arel, I., and Holleman, J. (2013). An analog online clustering circuit in 130nm cmos. In *Solid-State Circuits Conference (A-SSCC), 2013 IEEE Asian*, pages 177–180.

Lu, J., Young, S., Arel, I., and Holleman, J. (2014). 30.10 a 1tops/w analog deep machine-learning engine with floating-gate storage in 0.13 um cmos. In *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*, pages 504–505.

Lu, J., Young, S., Arel, I., and Holleman, J. (2015). 30.10 a 1tops/w analog deep machine-learning engine with floating-gate storage in 0.13 um cmos. *to appear in IEEE Journal of Solid State Circuits.*

MacQueen, J. B. (1967). Some methods for classification and analysis of multivariate observations. In Cam, L. M. L. and Neyman, J., editors, *Proc. of the fifth Berkeley*

Symposium on Mathematical Statistics and Probability, volume 1, pages 281–297. University of California Press. 9

MarcAurelio Ranzato, Y., Boureau, L., and LeCun, Y. (2007). Sparse feature learning for deep belief networks. *Advances in neural information processing systems*, 20:1185–1192. 24

Mishtal, A. and Arel, I. (2012). Jensen-shannon divergence in ensembles of concurrently-trained neural networks. In *Machine Learning and Applications (ICMLA), 2012 11th International Conference on*, volume 2, pages 558–562. 47

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition. 8

Ngiam, J., Chen, Z., Chia, D., Koh, P. W., Le, Q. V., and Ng, A. Y. (2010). Tiled convolutional neural networks. In Lafferty, J., Williams, C., Shawe-Taylor, J., Zemel, R., and Culotta, A., editors, *Advances in Neural Information Processing Systems 23*, pages 1279–1287. Curran Associates, Inc. 78, 79

Ngiam, J., Khosla, A., Kim, M., Nam, J., Lee, H., and Ng, A. Y. (2011). Multimodal deep learning. In *International Conference on Machine Learning (ICML)*, Bellevue, USA. 3

Pearson, K. (1901). *On Lines and Planes of Closest Fit to Systems of Points in Space*. University College. 19

Pedrycz, W. (1998). Conditional fuzzy clustering in the design of radial basis function neural networks. *Neural Networks, IEEE Transactions on*, 9(4):601–612. 41

Ratha, N. K., Chen, S., and Jain, A. K. (1995). Adaptive flow orientation-based feature extraction in fingerprint images. *Pattern Recognition*, 28(11):1657 – 1672. 19

Reynolds, D. (2008). Gaussian mixture models. *Encyclopedia of Biometric Recognition*, 2(17.36):14–68. 12

Salakhutdinov, R. and Hinton, G. E. (2007). Learning a nonlinear embedding by preserving class neighbourhood structure. *Journal of Machine Learning Research - Proceedings Track*, 2:412–419. 77

Scherer, D., Schulz, H., and Behnke, S. (2010). Accelerating large-scale convolutional neural networks with parallel graphics multiprocessors. In *Artificial Neural Networks–ICANN 2010*, pages 82–91. Springer. 79

Schmidhuber, J. (2012). Multi-column deep neural networks for image classification. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, CVPR '12, pages 3642–3649, Washington, DC, USA. IEEE Computer Society. 77

Schmidhuber, J. (2014). Deep learning in neural networks: An overview. *CoRR*, abs/1404.7828. 2

Simard, P., Steinkraus, D., and Platt, J. (2003). Best practices for convolutional neural networks applied to visual document analysis. In *Document Analysis and Recognition, 2003. Proceedings. Seventh International Conference on*, pages 958 – 963. 75, 77

Street, W. N., Wolberg, W. H., and Mangasarian, O. L. (1993). Nuclear feature extraction for breast tumor diagnosis. In *SPIE's Symposium on Electronic Imaging: Science and Technology*, pages 861–870. International Society for Optics and Photonics. 19

Torralba, A., Fergus, R., and Freeman, W. (2008). 80 million tiny images: A large data set for nonparametric object and scene recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(11):1958–1970. 69

Tsividis, Y. and McAndrew, C. (2011). *Operation and Modeling of the MOS Transistor*. Oxford University Press. 49

Uykan, Z., Guzelis, C., Celebi, M., and Koivo, H. (2000). Analysis of input-output clustering for determining centers of rbfn. *Neural Networks, IEEE Transactions on*, 11(4):851–858. 41

van Rossum, M., O'Brien, B. J., O'brien, B. J., and Smith, R. G. (2003). Effects of noise on the spike timing precision of retinal ganglion cells. *Journal of Neurophysiology*, 89:2406–2419. 5

Wan, L., Zeiler, M., Zhang, S., LeCun, Y., and Fergus, R. (2013). Regularization of neural networks using dropconnect. In *Proc. International Conference on Machine learning (ICML'13)*. 78, 79

Young, S., Arel, I., Karnowski, T. P., and Rose, D. (2010). A fast and stable incremental clustering algorithm. In *Information Technology: New Generations (ITNG), 2010 Seventh International Conference on*, pages 204–209. 28

Young, S., Davis, A., Mishtal, A., and Arel, I. (2014a). Hierarchical spatiotemporal feature extraction using recurrent online clustering. *Pattern Recognition Letters*, 37:115–123.

Young, S., Lu, J., Holleman, J., and Arel, I. (2014b). On the impact of approximate computation in an analog destin architecture. *Neural Networks and Learning Systems, IEEE Transactions on*, 25(5):934–946. 49

Young, S. R. and Arel, I. (2012). Recurrent clustering for unsupervised feature extraction with application to sequence detection. In *Machine Learning and Applications (ICMLA), 2012 11th International Conference on*, volume 2, pages 54–55. IEEE.

Yuille, A. L., Hallinan, P. W., and Cohen, D. S. (1992). Feature extraction from faces using deformable templates. *International journal of computer vision*, 8(2):99–111. 18

Appendix

Appendix A

Ensembles Operating on Unique Subsets of DeSTIN Beliefs

Although using ensembles that operate on unique subsets of DeSTIN's belief states was not necessary as outlined in Section 3.4, some results are presented here. An ensemble of 11 MLP classifers as used in Chapter 5 is used. Each ensemble member operates on a random subset of the belief states. Figures A.1 - A.3 demonstrate the classification accuracy achieved on the three standard datasets as the size of the random subset of features provided to each classifier is varied. In these results, no increase in performance is seen by providing fewer features to each classifier as was seen when using simpler classifiers (decision trees) and domain specific features (Bryll et al., 2003). However, on the MNIST and CIFAR-10 tasks these results indicate an opportunity to decrease the number of features provided to each classifier and maintain comparable performance to using all features which would reduce the computational complexity of each classifier. On the PEMS-SF task, ensembles do not offer any improvement over a single network. This is a result of this dataset only containing 267 training examples, while the other datasets contain at least $50,000$ training examples.

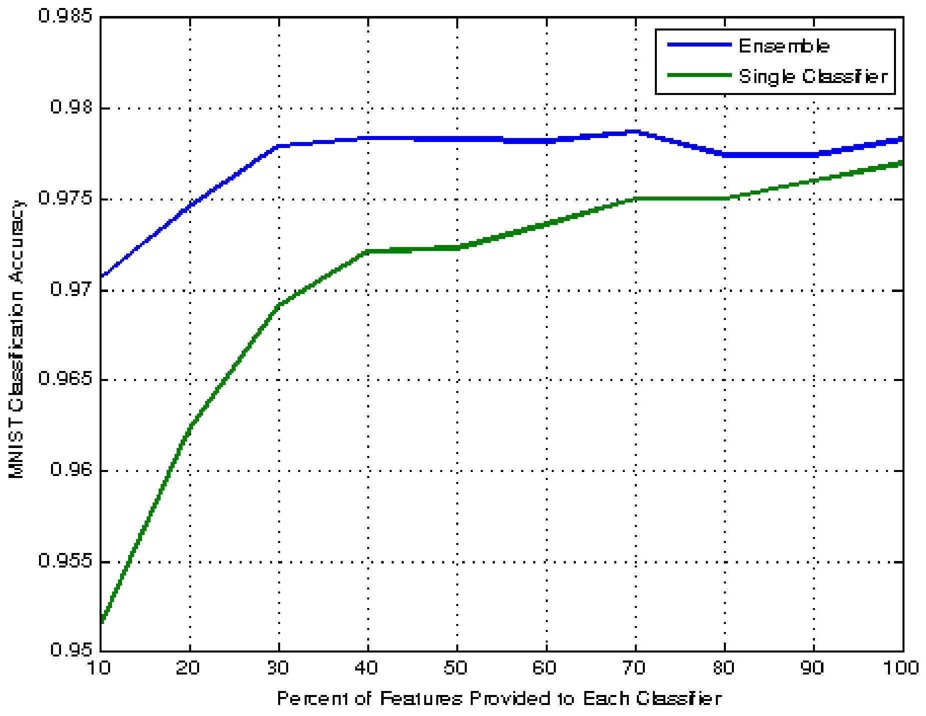


Figure A.1: MNIST: Classification accuracy versus percentage of features provided to each classifier

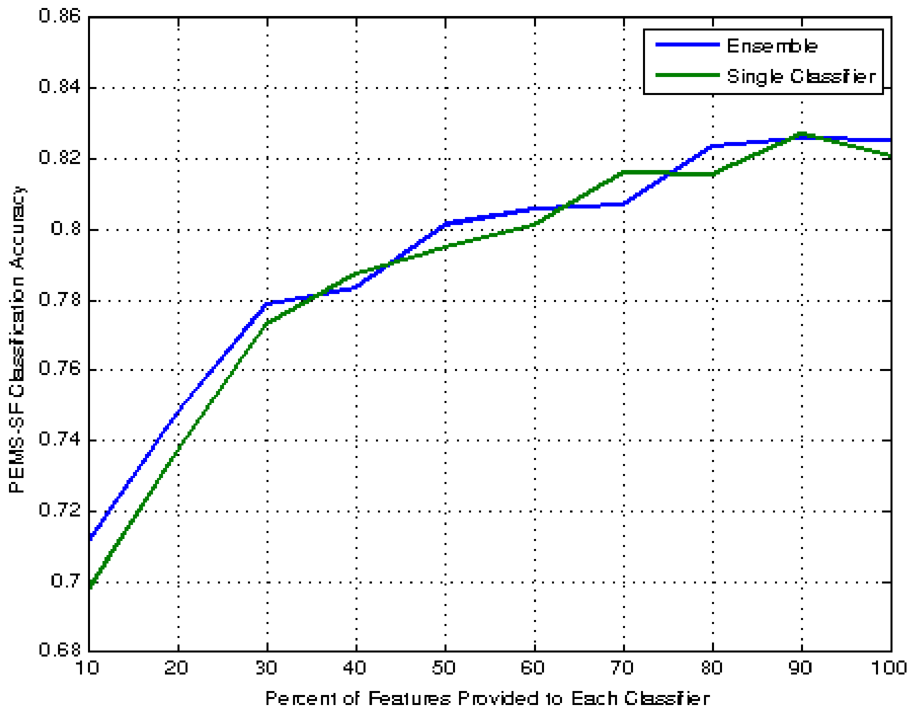


Figure A.2: PEMS-SF: Classification accuracy versus percentage of features provided to each classifier

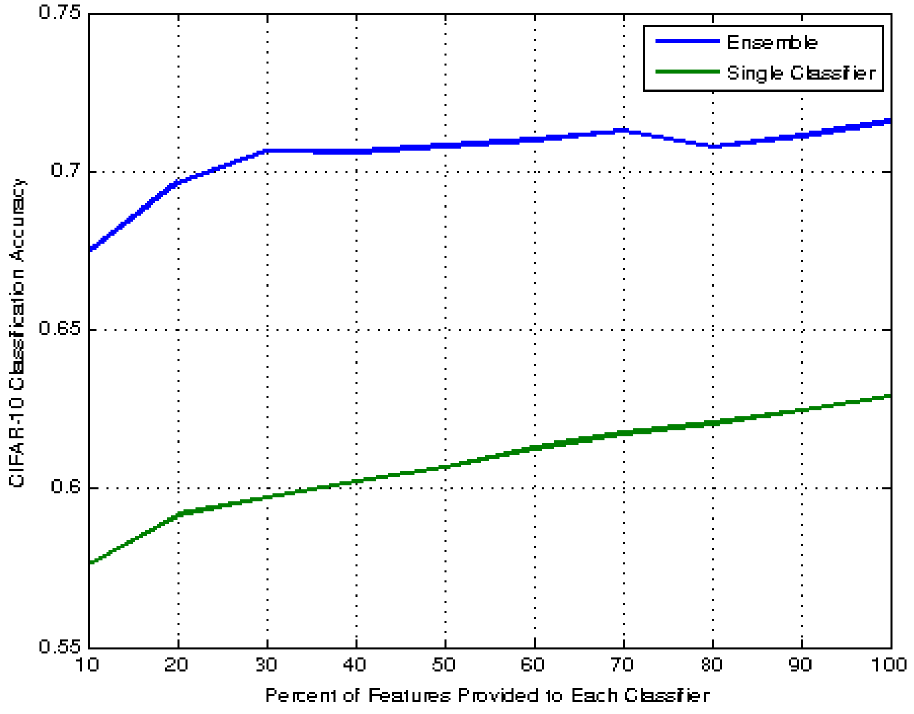


Figure A.3: CIFAR-10: Classification accuracy versus percentage of features provided to each classifier

Vita

Steven Robert Young was born in Flint, Michigan in 1986 to parents Jay and Teresa Young. In 2005 he graduated as the valedictorian from Marshall County High School in Lewisburg, Tennessee. He received his Bachelor of Science in Electrical Engineering from the University of Tennessee in 2010. Steven began his pursuit of the PhD degree in Computer Engineering in 2010 at the University of Tennessee. His graduate studies were supported by the J. Wallace and Katie Dean Graduate Fellowship. He has worked as a teaching assistant and received the Outstanding Graduate Teaching Assistant award from the Electrical Engineering and Computer Science department in 2013. He has also worked as a research assistant in the Machine Intelligence Lab supporting multiple funded projects. He began working in the Machine Intelligence Lab at the University of Tennessee as an undergraduate in 2008 and has participated in several summer research opportunities at Oak Ridge National Laboratory. He graduated with a Doctorate of Philosophy in Computer Engineering in December 2014 and has accepted a post-doctoral position at Oak Ridge National Laboratory. He has been married to Angela Young since 2011.