Dynamical System Autoencoders

Shiquan He*, Randy Paffenroth†, Olivia Cava‡, Cate Dunham§
Mathematical Sciences†, Computer Science†, and Data Science†‡§*
Worcester Polytechnic Institute
Worcester, MA 01609, USA
she5@wpi.edu*, rcpaffenroth@wpi.edu†, okcava@wpi.edu‡, cmdunham@wpi.edu§

2024 International Conference on Machine Learning and Applications (ICMLA) | 979-8-3503-7488-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICMLA61862.2024.00231

***Abstract*—Autoencoders represent a significant category of deep learning models and are widely utilized for dimensionality reduction. However, standard Autoencoders are complicated architectures that normally have several layers and many hyper-parameters that require tuning. In this paper, we introduce a new type of autoencoder that we call dynamical system autoencoder (DSAE). Similar to classic autoencoders, DSAEs can effectively handle dimensionality reduction and denoising tasks, and they demonstrate strong performance in several benchmark tasks. However, DSAEs, in some sense, have a more flexible architecture than standard AEs. In particular, in this paper we study simple DSAEs that only have a single layer. In addition, DSAEs provide several theoretical and practical advantages arising from their implementation as iterative maps, which have been well studied over several decades. Beyond the inherent simplicity of DSAEs, we also demonstrate how to use sparse matrices to reduce the number of parameters for DSAEs without sacrificing the performance of our methods. Our simulation studies indicate that DSAEs achieved better performance than the classic autoencoders when the encoding dimension or training sample size was small. Additionally, we illustrate how to use DSAEs, and denoising autoencoders in general, to perform supervised learning tasks.**

***Index Terms*—autoencoder, dynamical system autoencoder, dimensionality reduction, sparse autoencoder, denoising, stacked autoencoder, supervised autoencoder**

I. Introduction and Related Work

Autoencoders (AEs) [1] [2] [3] are an important deep learning architecture that contains encoder and decoder parts. The encoder transforms raw input into a lower-dimensional representation that can be utilized for other tasks, effectively serving as a dimensionality reduction process. The decoder reconstructs the encoded representation back to the original raw dimension, aiming to minimize the disparity between the raw input and the reconstructed output. While the bulk of the research on AEs revolves around a single AE being applied to data, our work revolves around considering AEs as iterative maps. Such a perspective leads to several important advances in AE architectures, training, and effectiveness. Our focus in this paper is on exploring iterative AEs.

While we are not the first to consider iterative AEs, we have been inspired by the idea of stacked autoencoders [4] and the work in [5] which suggested a close connection between AEs and dynamical systems and demonstrated the effectiveness of *iterating AEs*. Our work can be considered as a simplification and distillation of the above ideas, in that we consider quite simple AE architectures, in fact with only a single trainable layer, that are designed from their inception to be used in an iterative training framework. A dynamical system is defined as one in which a function describes how the system evolves over time, and dynamical systems have a broad theoretical foundation [6]. In particular, the Dynamical System Autoencoder (DSAE) is a new type of autoencoder architecture that we propose. Its iterative process is a discrete dynamical system, and this is the reason why we name it a DSAE. DSAEs employ the same function iteratively in both the encoding and decoding processes, which differentiates them from classic AEs. We provide a detailed explanation of the DSAE architecture in the Methodology section.

Our work extends the foundational work in [5] in several important directions. In particular, the method in [5], while quite important and interesting, iterates standard AE architectures. *Our proposed method is a discrete iterative dynamical system from its very inception.* Our goal of this paper is to propose a new deep learning architecture, without focusing on the theoretical framework, which we will leave for future work. However, we do observe that connections between deep learning and dynamical systems are gaining attention in recent years [7], [8], [9]. Deep neural networks can be viewed as discrete nonlinear dynamical systems, in which a linear map followed by a nonlinear activation at each step consist the basic dynamics [8]. [7] and [9] explored deep neural networks through continuous dynamical systems and investigated them from approximation theory perspective. These research articles provide valuable insights into the theoretical framework for understanding deep learning.

Recent research indicates that when AEs are overparameterized–specifically, when they have deep and wide layers–they can memorize training instances through saving them as attractors in the learnt function. The attractive property of AEs also extends to other overparameterized deep neural networks [5]. Through iterating the trained model, the training examples can be recovered. This finding in [5] connects overparameterized AEs with dynamical systems. However, it is important to note that the learnt model itself does not constitute a dynamical system. Additionally, their discussion primarily focused on overparameterized AEs and neural networks. In our proposed method, the function may utilize deep neural networks, but is not necessarily very deep. There are also studies on stacked autoencoders and stacked denoising autoencoders [4] in which learnt autoencoders or denoising autoencoders are stacked into a bigger model. These types of models can capture hierarchical representations of data and are suitable for complex data patterns. Unlike these approaches, our proposal does not require training a sequence of autoencoder structures, which simplifies the training process. There was also research using iterative deep neural network architecture. [10] developed an iterative neural network in which a stacked denoising autoencoder and a recovery function was iterated. Their iteration process is similar to ours. However, their iterated function format is more specific. In contrast, the function format in our proposed method allows for greater flexibility. For example, it can be a single layer perceptron or multi-layer perceptron (MLP). Another difference is the way in which we implement the encoder and decoder.

1946-0759/24/$31.00 ©2024 IEEE
DOI 10.1109/ICMLA61862.2024.00231

We elaborate upon this in detail in the Methodology section.

Let us review the applications of AEs and prepare to introduce how DSAEs address these applications. AEs are an example of a technique that is commonly used for unsupervised learning – dimensionality reduction. They can achieve nonlinear dimensionality reduction, and accordingly offer advantages over traditional dimensionality reduction methods like classic principal component analysis (PCA) [11] that only performs linear transformation. Another important application of AEs is denoising. Namely, we can apply AEs to remove noise that is in the inputs. This type of AEs are called denoising autoencoders (DAEs) [7]. During training, the model receives a noised version of the input. The original raw input is used to compute the loss, which measures the difference between the reconstructed output and the raw input. Our proposed DSAEs can also be utilized for denoising tasks. For instance, we can utilize DSAEs to remove noise from images. In the Experiment section, we will demonstrate how the classic DAEs and denoising DSAEs perform under two noise conditions.

It is less widely recognized that AEs can also be used for supervised learning. In fact, in the sequel we will demonstrate for DSAEs that supervised and unsupervised problems lay on a continuum. As an illustration, a common use case is classification problems with not many labeled instances, which can hinder the training of effective deep learning models, as these typically require large sample sizes. However, it can be easier or cheaper to obtain relevant unlabeled data. We can pre-train an autoencoder from these unlabeled data and only take the encoder part for downstream task. As the encoded dimension is usually small, fine-tuning on top of that may require much smaller sample size so that supervised learning tasks can be performed [12]. In addition to using them as pre-trained encoders, variations of AEs can also be used for supervised learning tasks. For example, predicting both inputs and targets jointly [13]. Similar to classic AEs, DSAEs can be applied to supervised learning tasks. Aside from being used as pre-trained models, they can be trained to perform supervised learning tasks. In the Methodology section, we explain how to use them to do unsupervised and supervised simultaneously as multi-task learning.

AEs feature simple architectures that are easy to implement and often exhibit robust performance. However, they are not without limitations. In Section III we will demonstrate that when the encoded dimension is very low or sample size is small, the encoder may inadequately capture the raw input, resulting in suboptimal reconstruction by the decoder. In the Methodology section, we will briefly review classic AEs, then formally introduce DSAEs and explain how they can be applied to different tasks. Sparse DSAEs are also introduced in the Methodology section. In the Experiment section, we compared the performances of classic AEs and DSAEs under different tasks, and demonstrate that DSAEs outperformed AEs in simulations, particularly when the sample size or encoding dimension was small. Results will be presented in the Results and Discussion section. [1]

II. Methodology

Before formally introducing DSAEs and their variations, we briefly review classic AEs [1] [2] [3]. As shown in Fig. 1, AE architecture consists of two parts–encoder and decoder. The encoder part encodes input into a lower-dimensional encoded representation, and the decoder transforms the encoded representation back to the input dimension with the goal of reconstructing the input. Accordingly, if we denote $\mathbf{x}$ as the input and $\hat{\mathbf{x}}$ as the output, AEs aim to minimize the difference between $\mathbf{x}$ and $\hat{\mathbf{x}}$. The encoder and decoder can simply be MLPs or convolutional layers (that can also be viewed as MLP).

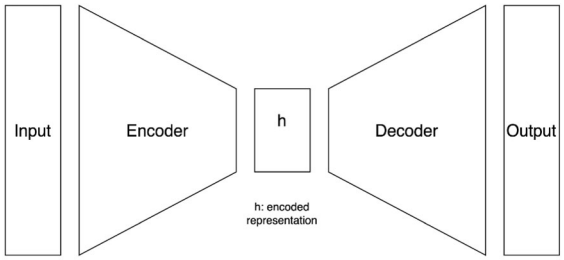

Fig. 1: Autoencoder architecture. The encoder and decoder have their only architectures. For example, they can be MLPs.

A. Dynamical System Autoencoders

Similar to classic AEs, there are both encoding and decoding processes in DSAE architecture. The encoder encodes input into a lower-dimensional latent space, and the decoder reconstructs the input from the latent space. Let us define f as a function that maps an input to output with the same dimension as the input. f can be an MLP. For example, in our experiments on the MNIST digit dataset, we used a linear layer that maps input to output with the same dimension plus a $sigmoid$ activation function; in the fashion MNIST dataset experiments, we use a linear layer plus a LeakyReLu activation. Let's assume the hidden dimension is h, and denote $\mathbf{x}$ as the flattened input and $\mathbf{0}$ as a vector with all zeros. Specifically, $\mathbf{0}$ in the left of (1) is a $h-$dimensional vector and $\mathbf{0}$ in the right of (1) has the same dimension as $\mathbf{x}$. We do not specifically denote the dimension of $\mathbf{0}$s and assume that their dimension are properly given so that the computation is appropriate. Then the encoding process is

$$f\left(f\left(\cdots f\left(f\left(\begin{bmatrix}\mathbf{x}\\ \mathbf{0}\end{bmatrix}\right)\right)\right)\right)=\begin{bmatrix}\tilde{\mathbf{x}}\\ \mathbf{h}\end{bmatrix}\rightarrow\begin{bmatrix}\mathbf{0}\\ \mathbf{h}\end{bmatrix} \tag{1}$$

where $\mathbf{h}$ is the encoded representation of the input. $\tilde{\mathbf{x}}$ is an intermediate that will be zeroed as shown in (1). It is important to note that this step is essential. The step where the vector $\begin{bmatrix}\tilde{\mathbf{x}}\\ \mathbf{h}\end{bmatrix}$ is tranformed to $\begin{bmatrix}\mathbf{0}\\ \mathbf{h}\end{bmatrix}$ plays the essential role of the information bottleneck produced by the encoder in an autoecoder.

In the decoding process, we apply the same function f iteratively on the output of the encoder

$$f\left(f\left(\cdots f\left(f\left(\begin{bmatrix}\mathbf{0}\\ \mathbf{h}\end{bmatrix}\right)\right)\right)\right)=\begin{bmatrix}\hat{\mathbf{x}}\\ \cdot\end{bmatrix} \tag{2}$$

where $\hat{\mathbf{x}}$ is the reconstructed input and the dot has the same dimension as $\mathbf{h}$ but the values are not needed for calculating loss or reconstructing input. The DSAE architecture is summazried in Fig. 2. The number of iterations in the encoding and decoding processes can be different. In our experiments, we set them to be the same.

[1] All code for producing the results in the paper are provides on the github.com repository https://github.com/squanhe/iterative-nn

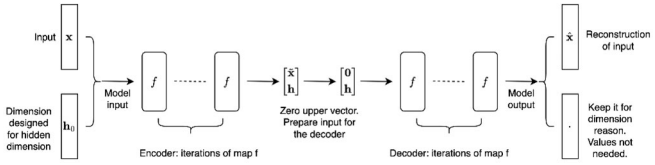

Fig. 2: Dynamical System Autoencoder architecture. In the model input, $\mathbf{h}_0$ has the same dimension as the hidden dimension $\mathbf{h}$, and a constant vector $\mathbf{0}$ is a good value for it. But if some prior information is available that you want to give to $\mathbf{h}_0$, the model architecture allows that.

From the above definition, we can discern the differences between typical neural networks or classic AEs and DSAEs. In a typical deep learning model, the model is a composition of functions for different layers, denoted as $f_L \circ \cdots f_2 \circ f_1(\mathbf{x})$ where L is the number of layers and $f_i, i = 1, \ldots, L$ are maps between layers. Accordingly, each layer map would have a set of parameters. In contrast, we use the same function f for all layers (more accurately, iterations) in DSAEs. For classic AEs, normally the encoder and decoder have different sets of parameters. However, we use the same function in a DSAE. Through iterations of the same function, it is also a composition of simple functions. Secondly, different layers of a deep learning model can have different output sizes. The only requirement is that the output size of a layer needs to match the input size of the next layer. In DSAEs, input and output sizes of f are the same and obviously are the same across iterations.

B. Adding Additional Zero-ed Input Dimensions into Input Vector

Our idea is to add some additional zero-ed input dimensions into the input vector with the goal of constructing more complex networks so that the new type autoencoder can perform even better. Additionally, we aim to test if adding additional zero-ed input dimensions into the input vector, while simultaneously reducing the hidden dimension size, can help to reach to a good reconstruction result. The mathematical thinking behind adding additional dimensions into the input is illustrated below. Let's assume f is one-layer perceptron, σ is the activation function, and denote

$$\begin{aligned} f\left(\begin{bmatrix}\mathbf{x}\\ \mathbf{0}\\ \mathbf{0}\end{bmatrix}\right) &= \sigma\left(\begin{pmatrix} W_{xx} & W_{xh} & W_{xj}\\ W_{hx} & W_{hh} & W_{hj}\\ W_{jx} & W_{jh} & W_{jj}\end{pmatrix}\begin{bmatrix}\mathbf{x}\\ \mathbf{0}\\ \mathbf{0}\end{bmatrix}\right)\\ &= \sigma\begin{pmatrix} W_{xx}\mathbf{x}\\ W_{hx}\mathbf{x}\\ W_{jx}\mathbf{x}\end{pmatrix} \end{aligned} \tag{3}$$

Then

$$\begin{aligned} &f\left(f\left(\begin{bmatrix}\mathbf{x}\\ \mathbf{0}\\ \mathbf{0}\end{bmatrix}\right)\right)\\ &=\sigma\left(\begin{pmatrix} W_{xx} & W_{xh} & W_{xj}\\ W_{hx} & W_{hh} & W_{hj}\\ W_{jx} & W_{jh} & W_{jj}\end{pmatrix}\begin{pmatrix}\sigma(W_{xx}\mathbf{x})\\ \sigma(W_{hx}\mathbf{x})\\ \sigma(W_{jx}\mathbf{x})\end{pmatrix}\right)\\ &=\sigma\begin{pmatrix} W_{xx}\sigma(W_{xx}\mathbf{x})+W_{xh}\sigma(W_{hx}\mathbf{x})+W_{xj}\sigma(W_{jx}\mathbf{x})\\ W_{hx}\sigma(W_{xx}\mathbf{x})+W_{hh}\sigma(W_{hx}\mathbf{x})+W_{hj}\sigma(W_{jx}\mathbf{x})\\ W_{jx}\sigma(W_{xx}\mathbf{x})+W_{jh}\sigma(W_{hx}\mathbf{x})+W_{jj}\sigma(W_{jx}\mathbf{x})\end{pmatrix} \end{aligned} \tag{4}$$

It shows from the red colored part $W_{hj}\sigma(W_{jX}X)$ that the additional dimension adds more complexity into the encoded dimension. Let's formally define the process.

$$f\left(f\left(\cdots f\left(f\left(\begin{bmatrix}\mathbf{x}\\ \mathbf{0}\\ \mathbf{0}\end{bmatrix}\right)\right)\right)\right)=\begin{bmatrix}\tilde{\mathbf{x}}\\ \mathbf{h}\\ \mathbf{j}\end{bmatrix}\rightarrow\begin{bmatrix}\mathbf{0}\\ \mathbf{h}\\ \mathbf{0}\end{bmatrix} \tag{5}$$

where $\mathbf{h}$ is the encoded representation of the input. Similar to (1), we do not need to worry about $\tilde{\mathbf{x}}$ and $\mathbf{j}$, as they will be zeroed out in the input of the decoder. For the decoding process

$$f\left(f\left(\cdots f\left(f\left(\begin{bmatrix}\mathbf{0}\\ \mathbf{h}\\ \mathbf{0}\end{bmatrix}\right)\right)\right)\right)=\begin{bmatrix}\hat{\mathbf{x}}\\ .\\ .\end{bmatrix}, \tag{6}$$

and $\hat{\mathbf{x}}$ is the reconstructed input. The two dots in (6) are only kept for the dimension and we do not need them for calculating loss.

C. Sparse Dynamical System Autoencoders

As f maps input to output with the same dimension, the DSAEs may require a higher number of parameters than classic AEs. This is not always the case, as we can define f in the format of MLP so that the number of parameters are comparable with classic AEs. We can reduce the number of parameters used in DSAEs and employ masked linear architecture [14] to implement the DSAEs. The idea is as follows: Assume that for normal linear layer, we have $\mathbf{y} = \mathbf{x}A^T + \mathbf{b}$ where $\mathbf{x}$ is the layer input, $\mathbf{b}$ is the bias term, A is the weight matrix, and $\mathbf{y}$ is the layer output. In masked linear architecture, A is expressed as $A = U \odot \Omega + W_0$ where W_0 is the initial weight, U is the trainable update and Ω is the mask (a matrix of boolean values). $\odot$ denotes the Hadamard product. When an element of Ω is 1, the corresponding parameter in U will be updated in each update step during training; if an element of Ω is zero, the corresponding parameter in U will not be updated during training process. Through this setup, we can let the weight matrix be sparse and control the number of trainable parameters. For simplicity, we refer to DSAEs implemented using the masked linear architecture as sparse DSAEs.

Specifically in our setup, W_{xx}, W_{xh}, W_{hx} and W_{hh} in (3) are sparse weight matrices. Namely, we randomly chose, say $p\%$ of the weight matrices as trainable parameters. In our case, $W_{xj}, W_{hj}, W_{jx}, W_{jh}$ and W_{jj}, are trainable. Of course, we can also make some of them sparse. The reason why we set the trainable parameters this way is that we want to see if adding additional dimensions as mentioned in (4) will help or not. More experiments on how to set weight matrices sparse is recommended, and we do not consider how we set them is optimal. Similarly, when f is a MLP, we can also apply the masked linear architecture to it.

D. Dynamical System Denoising Autoencoders

Dynamical system denoising autoencoders (DSDAEs) are DSAEs that are used for denoising tasks. Thus, the model architectures are the same as DSAEs. Sparse DSAEs can also be employed to control the number of parameters. Model output of DSDAEs are the reconstructed input that is expected to resemble the raw input without noise. Like DAEs, during training time, some noised version of input is given as the model input and the raw input is used to calculate model loss (calculating difference between model output and the raw input without noise). At inference time, a trained DSDAE is expected to denoise an input with noise.

E. Dynamical System Autoencoders for Supervised Learning Tasks

Like classic AEs, DSAEs can be utilized for supervised learning tasks. As mentioned in the Introduction, it is common that only a small number of labeled data is available while much more unlabeled data exit. In this scenario, we can train a DSAE using the unlabeled data, take the encoder part, and add additional layers on top of that for supervised learning. Other methods include the application of a classic machine learning model, such as SVM [15] or Xgboost [16], and use the encoded representation as the input for the machine learning model. Since there would be fewer parameters in need of training, the required sample size would be smaller. We can also fine-tune the encoder part as needed.

Alternatively, we can add supervised learning into the AE or DSAE architecture. Let's assume that we need to do some classification task. Denote $\mathbf{y}$ as the one-hot encoded true label and $\mathbf{y}_0$ as some constant vector, say a vector of 0.1s, that has dimension K, the same dimension as $\mathbf{y}$. Though a constant vector $\mathbf{y}_0$ is a good choice, some biased value vector where some positions have larger values than others can be appropriate if some prior information regarding label is available. Assume $\mathbf{x}$ is the raw input. Then we can use $\left(\mathbf{x}^T,\mathbf{y}_0^T\right)^T$ as an input to DSAE. Assume the output of the DSAE is $(\hat{\mathbf{x}},\hat{y})$, we would expect (or train the model so that) $\hat{\mathbf{x}}$ to be the reconstructed input and $\hat{y}$ to be the prediction of $\mathbf{y}$. The loss would be the weighted summation of reconstruction loss and prediction loss as following:

$$\begin{aligned} Loss &= L_{reconstruction}+\lambda L_{prediction} \\ &= \frac{1}{2n}\sum_{i=1}^{n}\|\hat{\mathbf{x}}_i-\mathbf{x}_i\|_2^2-\lambda\sum_{i=1}^{n}\sum_{k=1}^{K}y_{i,k}\log\hat{y}_{i,k} \end{aligned} \quad (7)$$

where i indicates the $i-$th instance, k indicates $k-$th element of encoded label or prediction, and λ is the weighting parameter. This is a classic example of multi-task learning [17]. However, in our case the DSAE form of a deep learning model makes such multi-task learning simple to implement. Fig. 3 illustrates the model architecture that we explained here. There is literature on using AEs for supervised tasks [13], [18], [19] and we leave them for readers to read.

It is important to note that for a DSDAE, the difference between supervised and unsupervised learning is merely a question of where the noise goes. As illustrated in (7), if we ignore the prediction loss of true target y, we have standard unsupervised learning. However, if we consider the input y_0 to the DSDAE (or DSAE) as a noisy guess for the true target y, then the reconstructed $\hat{y}$ is now a supervised prediction. The amount of noise in the input y_0 and the presence or absence of a loss term for the reconstruction of true underlying target y controls whether the problem is supervised or unsupervised.

III. Experiments

We used the MNIST (Modified National Institute of Standards and Technology) [20] and the fashion MNIST [21] datasets for our experiments. The MNIST data are commonly used for dimensionality reduction experiments and considered as a benchmark dataset. There are $60,000$ 28×28 images of hand written digits from 0 to 9 in the training set and $10{,}000$ images of digits in the test set. The training set was used for model training and the test set was used for model evaluation. Before final model training, the training set was split into a temporary training (80%) and validation (20%) sets for hyperparameter tuning. In order to show the usability of DSAEs, we also utilized the fashion MNIST data which is more complicated and not as popular as the MNIST data for dimensionality reduction tasks. Similar to the MNIST data, there are also $60{,}000$ 28×28 balanced labeled images in the training set and $10,000$ labeled images in the test set. The image labels are T-shirt/top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, and ankle boot. The fashion dataset can be used for classification tasks. In our experiment, we used this data for the dimension reduction task.

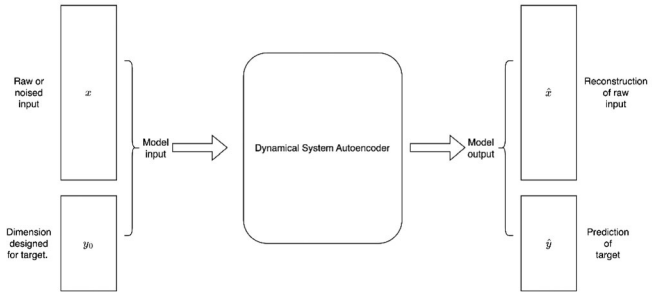

Fig. 3: Supervised Dynamical System Autoencoder training data organization. In the model input, there are two parts: one for raw or noised feature input and the other part y_0 is the dimension left for target, which can be either a constant vector or scalar or some vector value from prior information. When x is a noised version of raw feature input, then the model is supposed to learn to perform denoising task.

In the following experiments, all DSAE models are of function format, with one linear layer and a nonlinear activation layer. For the MNIST data, sigmoid was used, and for the fashion MNIST data, LeakyReLu was used.

For model evaluation, the test sets were used, respectively for both datasets. To obtain distributions of the model performance metrics, we set random seeds and accordingly generated 100 data loaders on the training sets. We trained the models on each of them and evaluated them on the test set of the corresponding task. Thus, 100 model performance metrics were obtained for each model on each task. We employed Wilcoxon rank-sum test to compare the distributions of performance metrics between two types of models and the p-values from the test were also provided. Mean squared error (MSE) loss was utilized for these experiments, except for the classification experiment, where we used a weighted combination of MSE loss and negative log loss.

A. Experiments on Dimensionality Reduction

As mentioned above, we are interested to see how AE and DSAE perform when the encoded dimension is small. We evaluated the model performances on dimensionality reduction task using the reconstruction loss – MSE loss. In our experiments, we specified the encoded dimension as 8 for the MNIST data and 16 for the fashion data. We trained a few models for the MNIST data as follows:

- Model 1: An AE with encoder layer sizes as $[784,128,64,32,8]$
- Model 2 A DSAE with hidden size as 8
- Model 3: A DSAE with hidden size as 8 and additional dimension size as 32

- Model 4: A sparse DSAE with hidden size as 8 and additional dimension size as 32 trained using masked linear architecture and randomly sampled 10% of above specified weights trainable

The number of parameters for each of the above models is presented in Table I. The training curves of the models from one training run (out of a total of one hundred runs) for the dimensionality reduction task on the MNIST data are displayed in Fig. 4. Some random samples from the test set are illustrated in Fig. 5.

Similarly, we performed experiments and trained a few models for the fashion data. The main differences between these models and those trained on the MNIST data are the encoding dimension and the additional dimension utilized. For the fashion data, the hidden dimension used was 16 and the additional dimensions used was 64. The model performance metrics are summarized in Table II. Some random examples from the test set are presented in Fig. 8.

B. Experiments on Dimensionality Reduction Using Small Training set

An interesting topic for us to investigate was to compare how the models perform under a small sample size training set. This experiment was conducted on the MNIST data. Only 6,000 randomly sampled instances from the training set were used, and the original test set was used for model evaluation. The same models as listed above were compared here. The results of this experiment are presented in Table III.

C. Experiments on Denoising Images

The MNIST data was also employed for denoising experiments. We considered two noised image setups. Let's denote an normalized image as $I_{i,j}, i = 1,\ldots,28, j = 1,\ldots,28$. For simplicity, the image index is ignored here, but by default, there is an image index to differentiate different images.

a) Gaussian Noises: Gaussian noised image is defined as

$$I'_{i,j} = I_{i,j} + \lambda \times \epsilon_{i,j}, \text{ where } \epsilon_{i,j} \sim \mathcal{N}(0,1).$$

In our experiment, we generated random noise from a standard normal distribution and set $\lambda = 0.5$.

b) Random Zero Noises: Each pixel of an image is randomly zeroed out with a certain probability.

$$I'_{i,j} = I_{i,j} \times \mathbf{Bernoulli}(p)$$

In our experiments, we set $p = 0.5$.

We compared the performances of DAEs, DSDAEs, and sparse DSDAEs on these setups and their performances are shown in Table IV. Model 1, 3 and 4 architectures were utilized here.

D. Experiment on Supervised Learning

Due to time constraints, we did not conduct a thorough experiment on the supervised learning task. We mainly aimed to demonstrate how to do it. One classifier was trained using the previously described approach on supervised learning tasks under the Gaussian noise setup. The model architecture used was similar to Model 3 but the dimension of the label was added. Non-informative prior was used for $\mathbf{y}_0$. Namely, $\mathbf{y}_0 = (0.1, 0.1, \ldots, 0.1)^T$. The prediction loss weight λ in (7) used for model training was 0.008. Fig. 12 presents some random examples from the test set, along with the predicted labels.

IV. Results and Discussion

Fig. 4 illustrates the training curves for models 1 to 4 on the dimensionality reduction experiment on the MNIST data for one time run. We can see that the DSAE achieved the lowest training loss. For that time run, DSAE performed the best (test set MSE losses: classic AE loss = 0.0172, DSAE loss = 0.0153, DSAE with additional dimensions loss = 0.0166 and sparse DSAE loss = 0.0167) and all the dynamical system models outperformed than the classic AE. Model 4 that is the sparse DSAE has the smallest number of parameters; however its performance (test set MSE loss = 0.0169) was better than the classic AE (test set MSE loss = 0.0172). However, we do not want to make conclusions based on a single training run. As mentioned in the Methodology section, for each model, we generated 100 training data loaders using randomly sampled seeds, used each of them to train the models, and evaluated them on the same test set. As Table I shows, we did not observe an advantage of DSAEs over classic AEs. Additionally, we did not see that DSAE with additional added dimensions outperform DSAE. Several potential reasons may explain this. Firstly, maybe the dimension reduction task on the MNIST data was relatively easy and accordingly no more added complexity was needed. Secondly, though we performed hyperparameter tuning, the tuning grid we used may not have been broad enough, and consequently we might have chosen suboptimal hyperparameters. Fig. 5 displays some random examples from the test set and the reconstructions from models 1 to 4 using that one time trained models mentioned above.

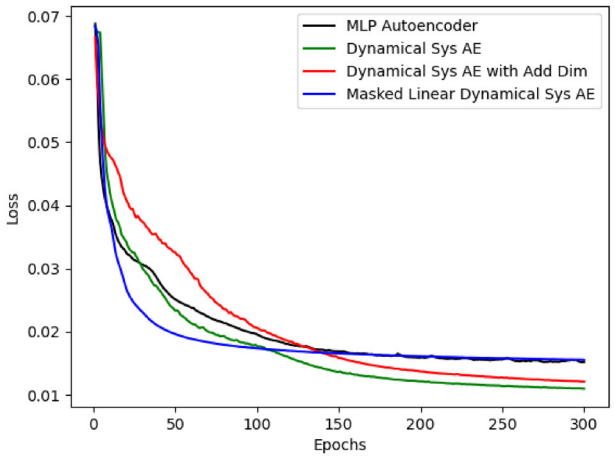

Fig. 4: Training curves on dimension reduction task on the MNIST data.

TABLE I: Model Performances on Dimension Reduction Task on the MNIST Test Set

Model	Num of Parameters	(MSE Loss$_{2.5}$, MSE Loss$_{97.5}$)
Model 1	222,936	(0.0166, 0.0181)
Model 2	628,056	(0.0160, 0.0185)
Model 3	679,800	(0.0166, 0.0213)
Model 4	**115,222**	(0.0163, 0.0185)

We also experimented on the dimensionality reduction task with the small training sample size on the MNIST data. Only

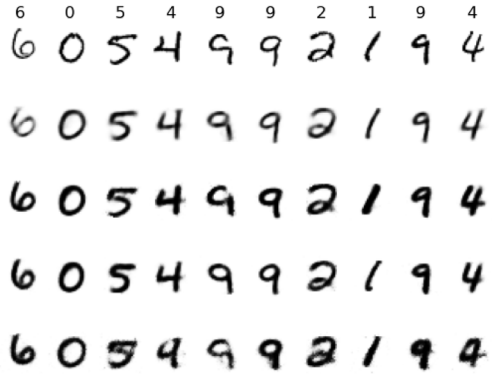

Fig. 5: Random examples from the MNIST test set and the reconstructions from the models for them. The first row lists the raw examples. The second row presents reconstructions from the trained AE. The third row shows reconstruction from the trained DSAE. The fourth row shows the reconstruction from DSAE with additional dimension input. The fifth row lists the reconstruction from the trained sparse DSAE.

10% of the training data was used, which is a harder problem than using the original training set. As shown in Table II, both the DSAE and sparse DASE outperformed the classic AE (p-values $= 1.262e^{-34}$ and $1.340e^{-34}$, respectively). Fig. 6 illustrates the distributions of MSE losses from the classic AE and sparse DASE models using the small training set, which demonstrates a statistically significant difference.

TABLE II: Model Performances on Dimension Reduction Task on the MNIST Data Test Set Using Small Training set

Model	Num of Parameters	(MSE $\text{Loss}_{2.5}$, MSE $\text{Loss}_{97.5}$)
Model 1	222,936	(0.0378, 0.0458)
Model 2	628,056	(0.0285, 0.0355)
Model 3	679,800	(0.0318, 0.0491)
Model 4	**115,222**	(0.0271, 0.0359)

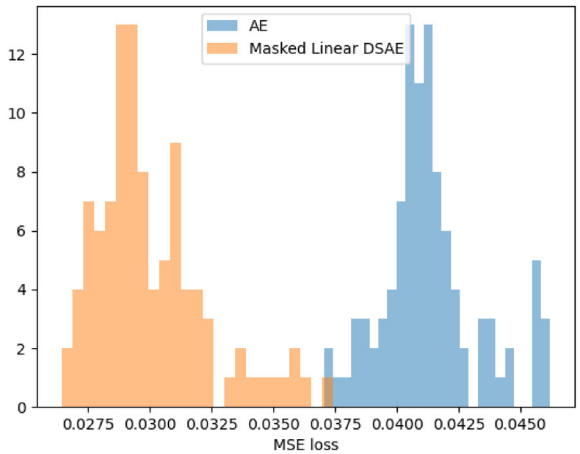

Fig. 6: Histograms of MSE losses of the classic AE and sparse DSAE models on the MNIST test set from 100 training runs using 6,000 training instances.

The dimension reduction experiment was also performed on the fashion data that have more detailed features in the images. This may represent a more challenging dimension reduction task than using the original MNIST data. As shown in Table III, all the dynamical system autoencoders performed significantly better than the classic AE (p-values $= 1.262e^{-34}$ for AE vs DSAE, $1.262e^{-34}$ for AE vs DSAE with additional dimensions, and $1.262e^{-34}$ for AE vs sparse DSAE, respectively). As an illustration, Fig. 7 shows the histograms of MSE losses from classic AE and sparse DSAE from the 100 training runs. There was no significant difference in MSE losses between the DSAE and the DSAE with additional dimension input (p-value $= 0.439$), which is different from our expectation. We showed that mathematically adding additional dimensions into the model incorporates more complexity into the model while keeping the same encoding dimension so accordingly may improve the model performance. However, empirically, we did not observe that. Fig. 8 displays random instances from the fashion test set and the reconstructions from the trained models 1 to 4 from one training run.

TABLE III: Model Performances on Dimension Reduction Task on the Fashion Data Test Set

Model	Num of Parameters	(MSE $\text{Loss}_{2.5}$, MSE $\text{Loss}_{97.5}$)
Model 1	223,456	(0.0131 0.0144)
Model 2	640,800	(0.0100 0.0106)
Model 3	747,360	(0.0099 0.0110)
Model 4	**171,280**	(0.0112 0.0118)

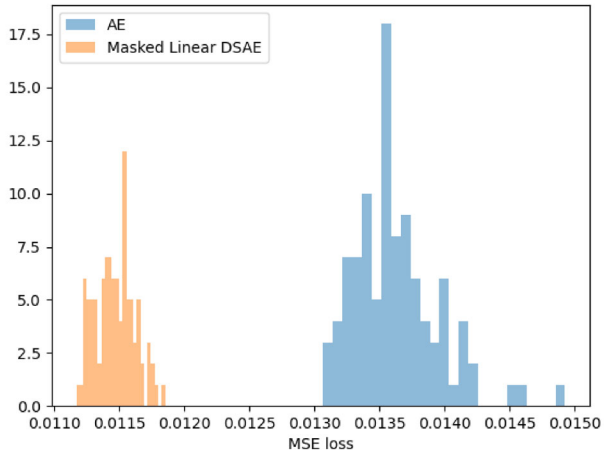

Fig. 7: Histograms of MSE losses of the classic AE and sparse DSAE models on the fashion test set from 100 training runs.

Table IV presents the performances of DAE, DSDAE, and sparse DSDAE in the denoising experiment on the MNIST data. Two types of noise are considered. DSDAE performed significantly better than DAE on both of the setups (p-value $= 1.601e^{-21}$ under the Gaussian noise setup and p-value $= 2.039e^{-34}$ under the random zero noise setup). Fig. 9 shows the distributions of MSE losses from those two types of models under the two noise setups. Here we admit that the two DSDAE models trained for those two setups had more parameters than the corresponding DAEs. Thus, a fairer comparison would involve assessing the performance of DAEs against a sparse version of DSDAEs. Due to time constraints, we were unable to conduct extensive hyper-parameter tuning on DSDAEs or adjust the architecture. In this experiment,

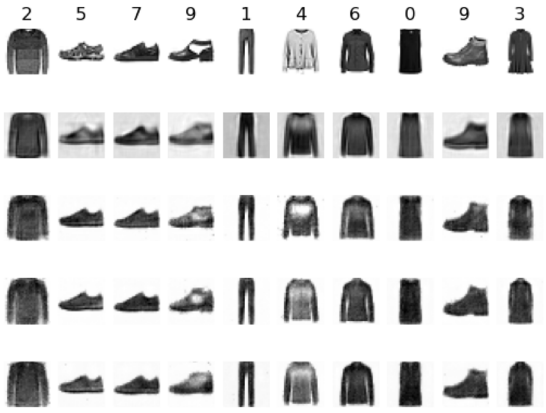

Fig. 8: Random examples from the fashion MNIST test set and the reconstructions from the models for them. The first row lists the raw examples. The second row presents reconstructions from the trained AE. The third row shows reconstruction from the trained DSAE. The fourth row shows the reconstruction from DSAE with additional dimension input. The fifth row lists the reconstruction from the trained sparse DSAE.

sparse DSDAEs did not show better performance than DAEs. Our primary objective was to introduce and explain of DSAE architecture. Fig. 10 and Fig. 11 display some random examples from the test set, the corresponding noised images, and the denoised images from one time training models.

TABLE IV: Model Performances on Denoising Task on the MNIST Data Test Set

Model	Noise Type	(MSE Loss$_{2.5}$, MSE Loss$_{97.5}$)
DAE	Gaussian	(0.0201, 0.0212)
DSDAE	Gaussian	(0.0178, 0.0264)
sparse DSDAE	Gaussian	(0.022, 0.0236)
DAE	Random 0s	(0.0175, 0.0675)
DSDAE	Random 0s	(0.0155, 0.0174)
sparse DSDAE	Random 0s	(0.0191 0.0202)

In the Methodology section, we also explained how to utilize DSAE models to perform supervised learning tasks. One model was trained on that under the Gaussian noise setup. The negative log loss of predicting the labels from the trained classifier on the test set was 1.499. More tuning can be performed on this experiment to improve the performance. We present only some initial results on the classification task to demonstrate how to do supervised learning tasks using DSAEs. Fig. 12 lists some random examples from the test set, their noised images, and the model output as the denoised images and the predicted labels.

V. Conclusions and Future Work

We proposed a new deep learning architecture, dynamical system autoencoder, which is capable of performing both unsupervised and supervised learning tasks, including dimensionality reduction, denoising, and target prediction. Our findings demonstrate that iterations of a simple function through both the encoder and decoder can perform the same tasks as what classic AEs can do. Through our experiments, we have shown that our proposed method may outperform classic AEs when the sample size or encoding dimension is small. Furthermore, we mathematically illustrated that adding additional dimensions into the input can increase the model's complexity without necessitating an increase in the encoding

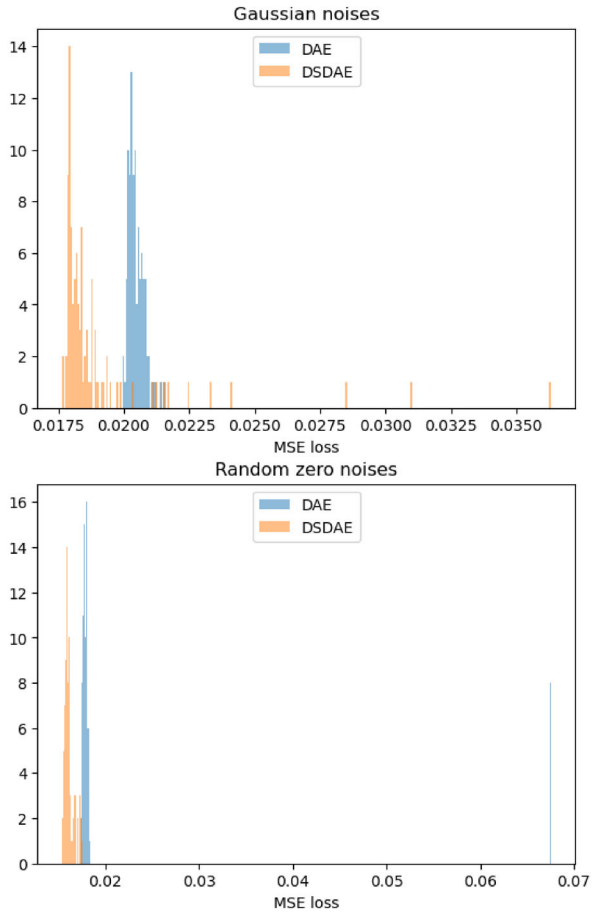

Fig. 9: Histograms of MSE losses of the denoising AE and DSDAE models on the MNIST test set.

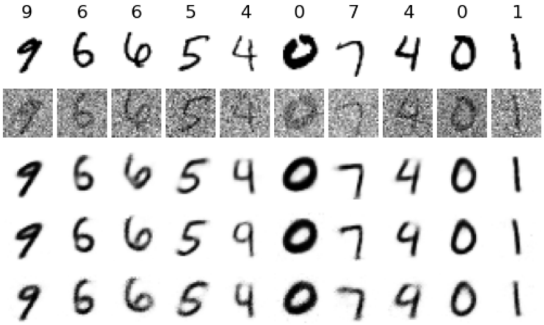

Fig. 10: Random examples from the MNIST test set, Gaussian noised images and the reconstructions from the denoising models. The first row lists the raw examples. The second row lists the Gaussian noised images. The third row shows denoised images from the denoising AE. The fourth row shows denoised images from the DSDAE. The fifth row shows denoised images from the sparse DSDAE.

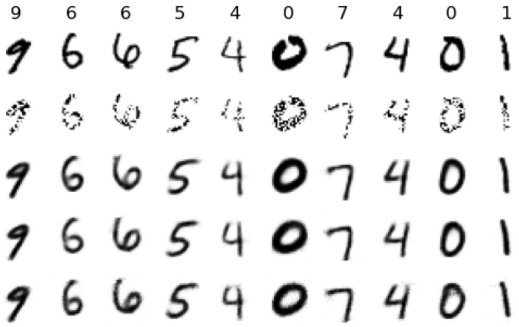

Fig. 11: Random examples from the MNIST test set, random zeroed images and the reconstructions from the denoising models. The first row lists the raw examples. The second row lists the zero noised images. The third row shows denoised images from the DAE. The fourth row shows denoised images from the DSDAE. The fifth row shows the denoised images from the sparse DSDAE.

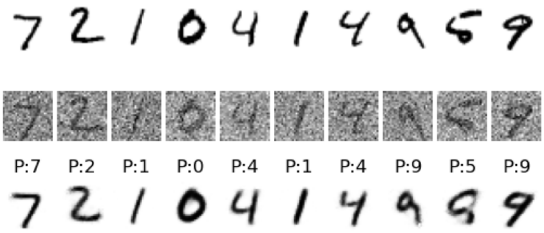

Fig. 12: Random examples from the MNIST test set, random Gaussian noised images and the reconstructions and predictions from the denoising DSAE. The first row lists the raw examples. The second row lists the Gaussian noised images. The third row shows denoised images from the DSDAE with the predicted labels denoted as P:*.

dimension. This may enhance the mode performance. There is considerable flexibility in function f used in DSAEs. For example, it can be an MLP or other structures, so long as the input and output of f are of equal size. For models with a large number of parameters, sparse, specifically masked linear, architecture can be employed to reduce the number of trainable parameters in a model. Well designed sparse architectures may also improve model performance as demonstrated in [14]. In summary, DSAEs represent a useful and innovative deep learning architecture.

Our experimental results indicate that DSAEs, implemented using iterations of a simple function f as a single-layer perceptron, can outperform classic AEs. Additionally, our findings suggest that sparse DSAEs, which utilize significantly fewer parameters, can achieve performance that is comparable to or even better than that of classic AEs. In future work, we aim to investigate on this simple DSAE architecture computationally and mathematically to understand the mechanisms behind its effectiveness.

ACKNOWLEDGMENT

The authors used Turing at Worcester Polytechnic Institute for model hyperparameter tuning and training, and would like to thank the Academic & Research Computing group at Worcester Polytechnic Institute for their support.

AUTHOR CONTRIBUTION STATEMENT

SH designed the study, implemented the work, and wrote the manuscript. RP designed the study, provided scientific direction, and contributed to the manuscript. OC and CD contributed to the final version of the manuscript.

REFERENCES

[1] M. A. Kramer, "Nonlinear principal component analysis using autoassociative neural networks," *AIChE journal*, vol. 37, no. 2, pp. 233–243, 1991.
[2] ——, "Autoassociative neural networks," *Computers & chemical engineering*, vol. 16, no. 4, pp. 313–328, 1992.
[3] P. Baldi, "Autoencoders, unsupervised learning, and deep architectures," in *Proceedings of ICML workshop on unsupervised and transfer learning*. JMLR Workshop and Conference Proceedings, 2012, pp. 37–49.
[4] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, and L. Bottou, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion." *Journal of machine learning research*, vol. 11, no. 12, 2010.
[5] A. Radhakrishnan, M. Belkin, and C. Uhler, "Overparameterized neural networks implement associative memory," *Proceedings of the National Academy of Sciences*, vol. 117, no. 44, pp. 27 162–27 170, 2020.
[6] R. L. Devaney, *A first course in chaotic dynamical systems: theory and experiment*. CRC Press, 2018.
[7] G.-H. Liu and E. A. Theodorou, "Deep learning theory review: An optimal control and dynamical systems perspective," *arXiv preprint arXiv:1908.10920*, 2019.
[8] E. Weinan, "A proposal on machine learning via dynamical systems," *Communications in Mathematics and Statistics*, vol. 1, no. 5, pp. 1–11, 2017.
[9] Q. Li, T. Lin, and Z. Shen, "Deep learning via dynamical systems: An approximation perspective," *Journal of the European Mathematical Society*, vol. 25, no. 5, pp. 1671–1709, 2022.
[10] Y. Zhang, R. Liu, S. Zhang, and M. Zhu, "Occlusion-robust face recognition using iterative stacked denoising autoencoder," in *Neural Information Processing: 20th International Conference, ICONIP 2013, Daegu, Korea, November 3-7, 2013. Proceedings, Part III 20*. Springer, 2013, pp. 352–359.
[11] I. T. Jolliffe and J. Cadima, "Principal component analysis: a review and recent developments," *Philosophical transactions of the royal society A: Mathematical, Physical and Engineering Sciences*, vol. 374, no. 2065, p. 20150202, 2016.
[12] W. H. L. Pinaya, S. Vieira, R. Garcia-Dias, and A. Mechelli, "Autoencoders," in *Machine learning*. Elsevier, 2020, pp. 193–208.
[13] L. Le, A. Patterson, and M. White, "Supervised autoencoders: Improving generalization performance with unsupervised regularizers," *Advances in neural information processing systems*, vol. 31, 2018.
[14] Q. Hershey, R. Paffenroth, and H. Pathak, "Exploring neural network structure through sparse recurrent neural networks: A recasting and distillation of neural network hyperparameters," in *2023 International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2023, pp. 128–135.
[15] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, 1995.
[16] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.
[17] L. Chen, A. J. Saykin, B. Yao, F. Zhao, A. D. N. I. (ADNI *et al.*, "Multi-task deep autoencoder to predict alzheimer's disease progression using temporal dna methylation data in peripheral blood," *Computational and Structural Biotechnology Journal*, vol. 20, pp. 5761–5774, 2022.
[18] J. Liu, C. Li, and W. Yang, "Supervised learning via unsupervised sparse autoencoder," *IEEE Access*, vol. 6, pp. 73 802–73 814, 2018.
[19] H. Sewani and R. Kashef, "An autoencoder-based deep learning classifier for efficient diagnosis of autism," *Children*, vol. 7, no. 10, p. 182, 2020.
[20] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.
[21] H. Xiao, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.