CHALMERS
UNIVERSITY OF TECHNOLOGY

Autoencoding As Regularization

A Neural Network Framework for Informative Data Representations

Master's thesis in Complex Adaptive Systems

LUKAS W. MERICLE

Department of Physics
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2019

Master's thesis 2019:NN

Autoencoding As Regularization

A Neural Network Framework for Informative Data Representations

LUKAS W. MERICLE

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2019

Autoencoding As Regularization
A Neural Network Framework for Informative Data Representations
LUKAS W. MERICLE

Supervisor: Dr. Robert Filman, PerformanceStar, LLC
Examiner: Professor Fabian Martin, Department of Electrical Engineering

Master's Thesis 2019:NN
Department of Physics
Chalmers University of Technology
SE-412 96 Gothenburg

Typeset in LaTeX
Gothenburg, Sweden 2019

Autoencoding As Regularization
A Neural Network Framework for Informative Data Representations
LUKAS W. MERICLE
Department of Physics
Chalmers University of Technology

Abstract

Machine learning relies on developing models which represent data in informative and simple ways. Taking inspiration from the subfield of multitask learning, we investigate the possibility of enhancing data representations at intermediate layers in a neural network. Specifically, we add a decoder layer whose task is to reconstruct the model's input from the intermediate representation. Along with this contribution, we introduce a number of algorithms for anomaly detection and supervised classification based on this framework and assess their performance. We find that anomaly detection works best in this framework when formulated as a classification problem between in-distribution and out-of-distribution data, and that supervised classification works best when using the simplest formulation with a linear classifier.

Keywords: autoencoder, regularization, multitask, representation, information, compression, anomaly detection

Acknowledgements

Thanks to Dr. Sukesh Patel for providing internship placement during the thesis project and to Dr. Robert Filman for acting as supervisor. Thanks are also due to Dr. Dominic Dotterer, Gabriel Kopito, and Joseph Fitch, all of whom participated in fruitful discussion about the ideas presented in this thesis as well as thesis document review. We acknowledge Prof. Fabian Martin for vetting and confirming that the idea has potential in industry and encouraging the direction of the thesis. Finally, thank you to my parents, James Mericle and Agnieszka Walczak, who provided me with the runway to be introduced to the advanced concepts which have been the focus of my studies and growth as a machine learning scientist.

LUKAS W. MERICLE, Santa Clara, CA, USA, July, 2019

Contents

List of Figures

List of Tables

1

Introduction

What makes machine learning tasks difficult is primarily the structure of the dataset in its native (original) space: correctly separating different classes or learning a regressor function over the space requires a model with highly complex structure for nontrivial datasets. The field of machine learning, and in particular deep learning, is largely concerned with learning models which are capable of transforming the data so that it is easier to perform the defined task. In recent years, this transformational and representational capacity has been achieved through model depth, that is, chaining many transformations together to achieve a final representation which makes tasks easy. Once an efficient representation of the data is found, the final step of classification or regression becomes likewise simple.

We seek to inform the optimization procedure for such models with the notion that *efficient representation of observations makes machine learning tasks trivial*. This will be the guiding principle of the following work: namely, that we train a model to perform one or more tasks while encouraging the model to use a representation which is as informative as possible. The latter goal is achieved by adding an autoencoding objective to the defined objective function, the former of which acts as a regularization to force the representation of the data to contain information about the dataset itself in addition to the task it must perform.

1.1 Motivation

The motivation behind this work is to design a system which efficiently represents data for various tasks. To achieve this, we can explicitly augment the objective function to discourage data representations which are lacking important information about the data.

1.1.1 Thought Experiment

To justify this approach, we consider a simple thought experiment, meant to elucidate the reasons that data representation is an important part of pattern recognition.

Description

A demon is the arbiter of a dataset $\mathcal{D} = \{(x, y) \mid x \in \mathcal{X}, y \in \mathcal{Y}\}$ and so knows which labels y correspond to which features x. This information is unavailable *a priori* to the experimenter. The experimenter wishes to accurately perform a task on the data she observes. Assume the demon can only communicate a set of features, *i.e.*, can send neither the explicit labels nor a function $f : \mathcal{X} \to \mathcal{Y}$ to the experimenter to fulfill her wish.

In general, it is not safe to assume that $\mathcal{X}$ is of sufficiently simple structure for the experimenter to learn accurate labels for the dataset. The demon must consider the experimenter's capacity for interpreting data, and the experimenter is constrained in the complexity of the function they can express. Therefore it is prudent to represent the data in as simple a way as possible for the experimenter.

Then the demon shall create a function $g : \mathcal{X} \to \mathcal{Z}$ which maps the original feature set into a compressed and informative representation $\mathcal{Z}$. The demon knows how the experimenter is likely to interpret the data and so defines g accordingly. In particular, the demon wants to organize the data such that i) the experimenter has no trouble partitioning the data according to their true labels, ii) the locations of the partition boundaries are obvious to the experimenter, and iii) relationships between points in $\mathcal{Z}$ reflect the relationships between corresponding points in $\mathcal{X}$. Without the last condition, the function would be trivial, mapping all data corresponding to label y to a single point z. With the last condition in place, the demon is forced to define a function which can generalize so that unseen data is treated appropriately in $\mathcal{Z}$ and the partition boundaries remain valid without adjustment.

In general, data is easier to work with when it has fewer dimensions; this is the inverse statement of the one characterizing the curse of dimensionality [1]. For the sake of the experimenter, it is prudent to define g such that $|z| << |x|$.

If the demon has done its job correctly, the function $h : \mathcal{Z} \to \mathcal{Y}$ subsequently defined by the experimenter can be almost arbitrarily simple and her wish is fulfilled with little effort.

Interpretation

In the thought experiment above, the demon plays the role of an *optimal informative embedding function* which organizes the data into some embedding space to make the task easy. The experimenter is then a subsequent network with enough capacity to perform its task using the embedding space but not the original space as its input.

Optimization of the embedding function proceeds by minimizing a loss functional containing information about the quality of the representation. The demon's knowledge of the capacity and skill of the experimenter may then correspond to augmenting the loss functional with an additional objective that encourages desirable properties of the representation with respect to the experimenter's classification accuracy. We can achieve this in practice by optimizing both objectives simultaneously.

2

Theory

2.1 Data Representation

2.1.1 Neural Network Autoencoders

Autoencoders are systems which learn the identity function, $x \mapsto x$. To create an autoencoder using neural networks, we define two networks, $\phi : \mathcal{X} \to \mathcal{Z}$ and $\psi : \mathcal{Z} \to \mathcal{X}$, named the encoder and decoder networks respectively, such that $(\psi \circ \phi)(x) \approx x \; \forall x \in \mathcal{X}$. The output of the encoder can be interpreted as a code or representation vector which carries information about the input to the decoder for subsequent reconstruction [2].

Typically, regularization is applied to the autoencoder either through the architecture of the network or through additional terms in the objective function. In the case of *undercomplete* autoencoders, the code layer (the layer between the encoder and decoder) has fewer nodes than the input, so learning the identity function is nontrivial because in general the input data cannot be losslessly represented by a smaller vector. The code layer then becomes a *representation* of the data in a smaller space, and the autoencoder can be said to perform nonlinear dimensionality reduction. If the encoder or decoder networks have excessive capacity, then the network will not learn the patterns which are indicative of the data's underlying distribution. Instead the network will overfit and will only reconstruct unseen data poorly, so constraining the network's architecture is an effective method of learning nontrivial, generalizable functions [3].

Definition 1. *The encoder half of an autoencoder should obey the condition*

$$x \approx y \to \phi(x) \approx \phi(y) \tag{2.1}$$

To encourage representations of the data which retain some meaning and generalizability, we usually require that the encoder constitutes a continuous mapping from the original space to the representative space.

The above condition defined by Definition 1 is required for appropriate inference on unseen data, because if this condition is not fulfilled, small changes in the input can propagate through the network, leading to an inconsistent notion of "locality" in the embedding space. Without this locality, we cannot rely on the embedding to

remain faithful to the original data.

We enforce continuity in the embedding by explicitly enforcing continuity in the encoder function, which amounts to restricting our choice of activation functions to those which are at least C^0-continuous. In fact, all of the typical choices for activation function fit this requirement. Although C^0-continuity is sufficient, better approximation is achieved with C^k-continuous functions in the limit as $k \to \infty$ [4]. Examples of activation functions which are C^∞-continuous are the sigmoid, hyperbolic tangent, and softplus functions.

2.1.2 Information Content And Capacity In Neural Networks

We can be more rigorous about the quality of the representation by interpreting it in terms of its information content. Specifically, we are interested in the mutual information between layers in the network, with the input and output layers being of prime importance. In the context of autoencoders, we are also interested in the mutual information between input and code layers, and perhaps also the mutual information between code and output layers.

Mutual information is the Kullback-Leibler divergence between a probability distribution and the product of its marginal distributions. This means that mutual information represents the information gain when considering a probability distribution over the random variables of interest, relative to treating the random variables as completely independent. This implies non-negligible "structure" in the probability distribution that is not accounted for by the marginal distributions of each random variable alone.

We start with the data processing inequality (DPI) [5, 6], which states that, for a probabilistic graphical model specified by $X \to Z \to Y$,

$$I(X;Y) \leq I(X;Z) \tag{2.2}$$

where $I(X;Y)$ is the mutual information between random variables X and Y. Put another way, the DPI states that information is irrecoverable once it is lost during the transitions through the graph.

Considering this result in the context of autoencoders, this says that the mutual information between the input and code layers is bounded from below by that between the input and output layers.

When training an autoencoder, convergence is achieved when reconstruction error is minimized. The theoretical global minimum of the reconstruction error is zero, when the outputs of the network correlate perfectly with the inputs. Then there is no information loss between input and output, and following the inequality in 2.2 we conclude that the autoencoder performs *lossless compression at the code layer*! Thus, the global optimum is the same for the objectives of maximizing mutual information and minimizing reconstruction error.

An idea closely related to the DPI is the Infomax principle [7], which states that a learning system undergoing optimization moves toward a state where the mutual information between input and output is maximized. Tishby et al. took this idea further with the "information bottleneck" hypothesis [8] which interprets neural network optimization as a tradeoff made at each layer in the network between representation complexity versus the input and retained information about the expected (true) output. This framework admits a natural interpretation where information content at the output of the network corresponds to the level to which the model can represent the necessary information about the training dataset.

2.2 Classification

The problem of classification is to segment the input domain into a number of regions – potentially overlapping and not necessarily simply connected – each corresponding to a unique class, such that all locations in a region represent inputs which belong to or are assigned a label corresponding to that class.

In the case of non-overlapping regions, *i.e.*, a partition over the input space, the task is called *multi-class classification*, whereas when the regions are allowed to overlap, the task is *multi-label classification*. The former corresponds to mutually exclusive labels, while the latter does not impose that constraint.

2.2.1 Neural Network Classifiers

The function implemented by a classifier built from a neural network will inherit the following useful properties of neural networks:

- it is defined over the entirety of the input domain
- its output is deterministic, meaning also that any boundaries defined by the output are a deterministic function of the network's parameters
- the activation function at the last layer can be chosen to model appropriate probability distributions

The last property warrants further discussion. The choice of activation function at the last layer in a neural network is constrained by the task that the network is assigned. For classification, we predict soft class assignments, so that all assignments can be interpreted as probabilities. The reason for this is twofold: first, hard class assignments provide no gradient information for cases where we want to optimize the network with a gradient descent algorithm; second, interpreting the neural network as a probabilistic model allows for powerful theory and analysis [2].

For multi-class classification, the softmax function is applied since it can model a draw from a multinomial distribution, corresponding to mutually exclusive outcomes. For multi-label classification, the sigmoid activation function can be used to

model a set of independent, simultaneous Bernoulli draws, corresponding instead to independent assignments for each class.

2.3 Multi-Task Learning (MTL)

The objective of MTL is to use a common, shared data representation to solve multiple tasks simultaneously [9]. MTL reduces model complexity by encouraging an informative representation that generalizes across disparate tasks. Requiring good performance on all the tasks can be seen as a form of regularization which adapts its influence based on the associated tasks.

Attempts to explicitly regularize the representation layer of the model, *i.e.*, by adding a regularization term to the loss function, are also present in the literature [10, 11, 12] and are usually formulated to induce sparsity in the features used for each task, to encourage the model to better separate shared features from unshared ones.

3

The learning system in detail

3.1 The System

There are a few equivalent perspectives to take on the proposed machine learning system, visualized in Figure 3.1. The first is of an autoencoder with additional task(s) such as classification being performed on the representation at the code layer. Another is as a vanilla classifier network with an additional autoencoder objective at a bottleneck layer in the network. Yet a third is as a multi-task learning system, where the shared body of the subnetworks is the encoder half of an autoencoder, and one head is the decoder half while the rest of the heads can pursue other tasks.

In all cases, a shared representation of the input data is learned at the interface between all the subnetworks, and this representation must support all the tasks which the networks have been assigned, most importantly being the autoencoding objective, which is given in the form

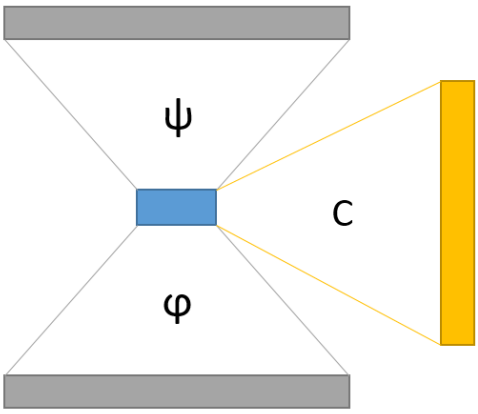
$$L_{AE}[f_\theta] = \frac{1}{N}\sum_i \|f_\theta(\vec{x}_i) - \vec{x}_i\|_2^2 \tag{3.1}$$

where $f_\theta(\vec{x}) = (\psi \circ \phi)_\theta(\vec{x})$ and N is the cardinality of the minibatch of data $\{\vec{x}\}_i$. This loss functional represents the usual mean-square error.

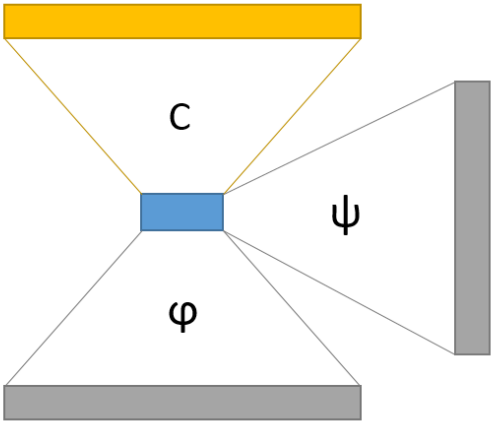
Assume we are developing a multi-task learning system with M subnetworks, each with an assigned task, excluding the autoencoding task. Let $L_k(\theta)$ be the loss function for each subnetwork indexed by k. Then the total loss function can be described as

$$L[f_\theta] = \sum_{k=1}^{M} L_k[f_\theta] + \lambda\, L_{AE}[f_\theta] \tag{3.2}$$

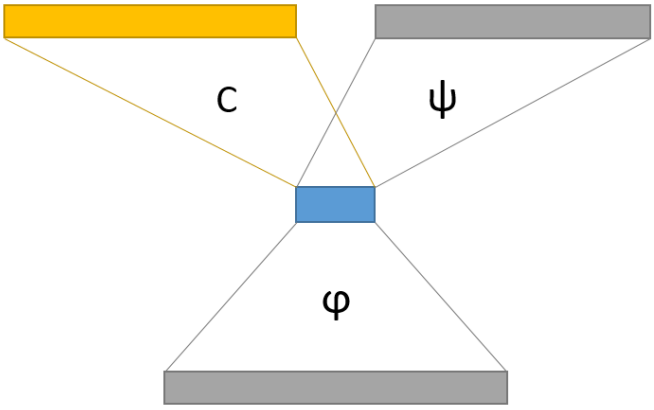
where λ is a hyperparameter scaling the influence of the autoencoding objective on the whole optimization process. Equation (3.2) takes the form of a regularized loss function, with the autoencoding objective acting as the regularizer. This perspective admits an interpretation where decoding the representation at some point in the network constrains the network to encourage informative representations there. Unlike other typical regularization methods, however, this regularization requires additional layers to be defined in the network and so increases the number of parameters being trained.



(a) Autoencoder plus classification objective



(b) Classifier plus autoencoding objective



(c) Multi-task learning system: classifier & autoencoder

Figure 3.1: The multiple perspectives on the addition of an autoencoding objective to a typical machine learning system. Here ϕ is the encoder half of the autoencoder, ψ is the decoder half, and C is the classification network. The input layer is at the bottom of each diagram. Gray boxes represent the data's features, and yellow boxes represent the data's labels. The blue box is the shared (bottleneck) layer.

Components Of The System

Although we frame the system as a neural network with an additional autoencoding objective, for the purpose of implementation it is more prudent to frame the system as an autoencoder with additional tasks being performed on the representation of the data at the code layer. That is, we define the architecture of the autoencoder separately from the tasks performed on the representation of the data at its code layer.

The autoencoder is defined to be as simple as possible, meaning its architecture is

symmetric about the code layer. We also construct the autoencoder simply, only specifying the number of hidden layers in each half of the autoencoder, the number of nodes in each hidden layer (being the same in all hidden layers), and the number of nodes in the code layer.

Then the tasks are each performed by simple neural networks, described by the number of layers and number of nodes per layer.

Later, in Section 3.2, we describe ways to use the code layer directly for anomaly detection, clustering, and classification, and evaluate these methods in Chapter 4.

The Vector-Quantization (VQ) Classifier

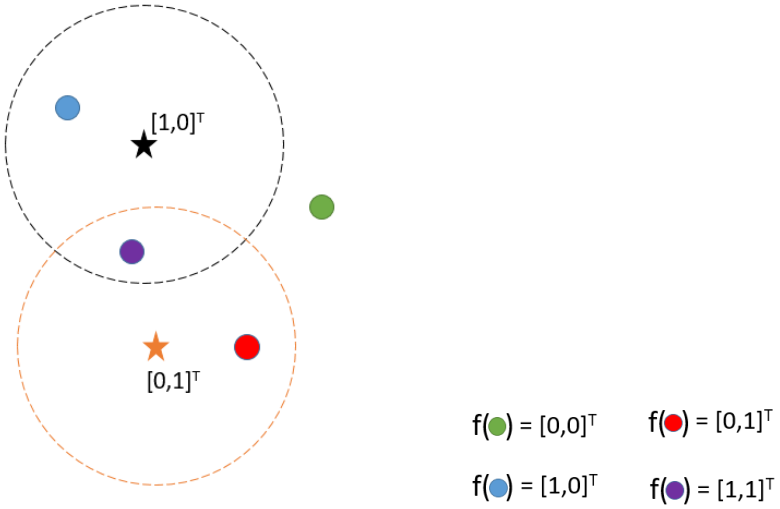


Figure 3.2: The VQ classifier classifies data based on whether it falls inside a decision radius, depicted here as dashed lines. The decision radius can be any positive value and specifies the radius of the associated label's decision sphere. The intersection of multiple decision spheres demarcates a region which is quantized into a class assignment vector.

First we define a new classifier architecture which classifies data points based on proximity to a class's *exemplar*. A visual aid is presented in Figure 3.2. The classifier assigns a label to the data by performing logistic regression on the norm of the vector difference between the data point and the exemplar. The logistic regression is parametrized by only two parameters, the decision radius b and the sharpness of the decision boundary w, and the prediction is governed by

$$\hat{y}_c = \sigma(-w\|\vec{x} - \vec{c}\|_2 + b) = \frac{1}{1 + \exp\left(w\|\vec{x} - \vec{c}\|_2 - b\right)} \tag{3.3}$$

where $\hat{y}_c$ is the predicted probability that the data represented by $\vec{x}$ is associated with the label represented by the exemplar $\vec{c}$.

We write Equation (3.3) in that form and add the constraint that w and b remains positive so that the decision boundary exists and the predicted value is greater than $1/2$ when inside the decision boundary. With these conditions the VQ classifier only represents viable clusters and the predictions it generates are sensible with respect to the clusters created.

In practice, we can achieve this by learning w' and b' as unconstrained free parameters and computing $w = \exp(w')$ and $b = \exp(b')$ at prediction time. As usual, we optimize with cross-entropy loss.

This classifier inherits its name from vector quantization, which is a technique for summarizing a distribution of points in a vector space by means of a small number of prototype vectors. This technique induces a map from each point to the prototype which best represents it, akin to unsupervised clustering. Notable examples of vector quantization in machine learning are the k-means and k-medoids algorithms, where each data point is represented by the centroid of its associated cluster.

3.2 Approaches for Solving Problems with the System

3.2.1 Anomaly Detection

Anomaly detection is a task characterized by a large collection of unlabeled data, and whose objective is to identify outliers or anomalous data points for review and, if necessary, subsequent action. Examples of problem domains for anomaly detection are: cybersecurity, to identify uncommon usage patterns and malicious activity; disasters, *e.g.*, earthquake prediction, to detect precursors to catastrophic events; manufacturing, to predict if a machine is behaving abnormally and requires maintenance; etc.

AD1

As a baseline, we perform anomaly detection using a simple technique which uses reconstruction error as a metric for anomalousness. The process is simple: train an autoencoder on the data, and define (or learn, if the labels are available) a threshold of reconstruction error above which the data is considered anomalous.

AD2

For the next method, depicted in Figure 3.3, we assume that when data is mapped closer to a cluster's center, the patterns expressed in that data are more common, since it is on average close to all the other data. To this end, we optimize the network by using the usual autoencoder objective, but with an additional penalty

corresponding to the square of the vector norm of the data,

$$L[f_\theta] = \frac{1}{N}\sum_{i=1}^{N} \|\phi_\theta(\vec{x}_i)\|^2 + \lambda\, L_{AE}[f_\theta] \tag{3.4}$$

The hope is that over time, the data representing more common patterns will be pushed toward the center while the data representing less common, *i.e.*, more anomalous, patterns will be further away. Then we can compute the anomaly score using the Mahalanobis distance of the data's embedding vector to the mean of the dataset's embedding vectors.

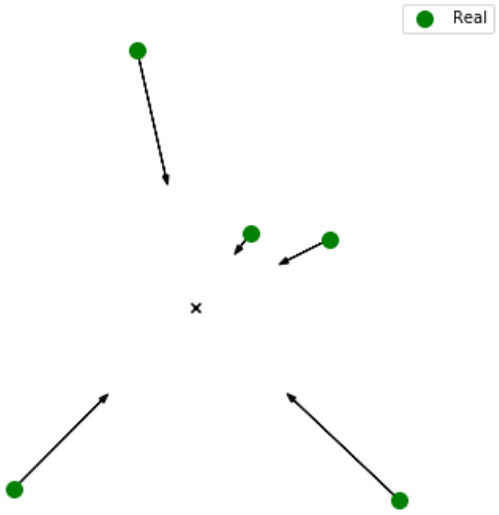


Figure 3.3: Visual representation of the gradients associated with an update using the AD2 method. Examples are depicted as dots, and the gradient vectors are depicted as arrows. The origin is depicted as '`x`'.

AD3

Here we leverage the smoothness of the embedding function $\phi_\theta(\vec{x})$ to "pull" anomalous data away from the origin. We achieve this by feeding in fake data and penalizing it more when it is closer to the origin, while encouraging real data to remain near the origin,

$$L[f_\theta] = \frac{1}{N}\sum_{i=1}^{N} \|\phi_\theta(\vec{x}_i)\|^2 + \frac{1}{N}\sum_{i=1}^{N} \|\phi_\theta(\vec{w}_i)\|^{-2} + \lambda\, L_{AE}[f_\theta] \tag{3.5}$$

where $\vec{w}$ represents the fake data (sampled from an assumed prior probability distribution). The anomaly score is computed as the Mahalanobis distance of the data's embedding vector to the mean of the dataset's embedding vectors.

From Definition 1 it follows that *anomalous data will be mapped closer to random inputs than data representing nominal behavior at the code layer of a trained autoencoder*. By this we mean that as training progresses, the autoencoder will learn to map fake data far away from the origin and real data closer to it. Data whose characteristics are observed less often will be more poorly represented in the autoencoder than data which exhibits more common behavior. If the real, but anomalous, data looks more like fake data, its location in the code space should be closer to the fake data, *i.e.*, further from the origin. We can exploit this property in anomaly detection by encouraging more-common patterns to be mapped to locations near the origin and less-common patterns to be mapped further away from the origin.

A visual aid is provided in Figure 3.4.

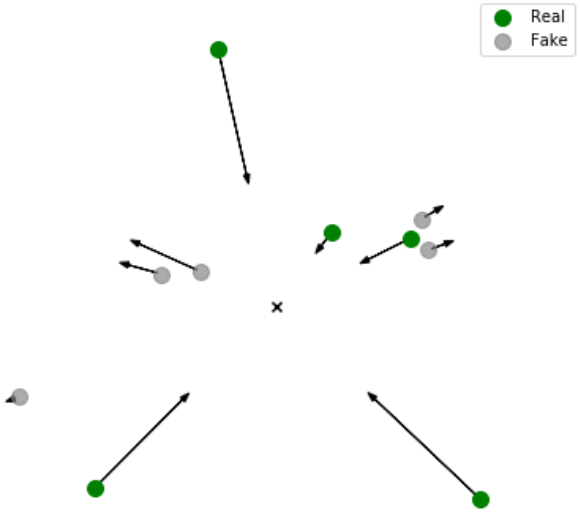


Figure 3.4: Visual representation of the gradients associated with an update using the AD3 method. Real examples are depicted as green dots, fake data as gray dots, and the gradient vectors are depicted as arrows. The origin is depicted as '`x`'.

AD4

Here we pose anomaly detection as a classification problem between the real data and fake, generated data. The classification is performed by our VQ classifier, and the exemplar vector is fixed to be the origin. We add a penalty onto the magnitude of the radius of the VQ classifier to ensure that the data's representation remains compact and near the origin. See Figure 3.5 for a visual aid.

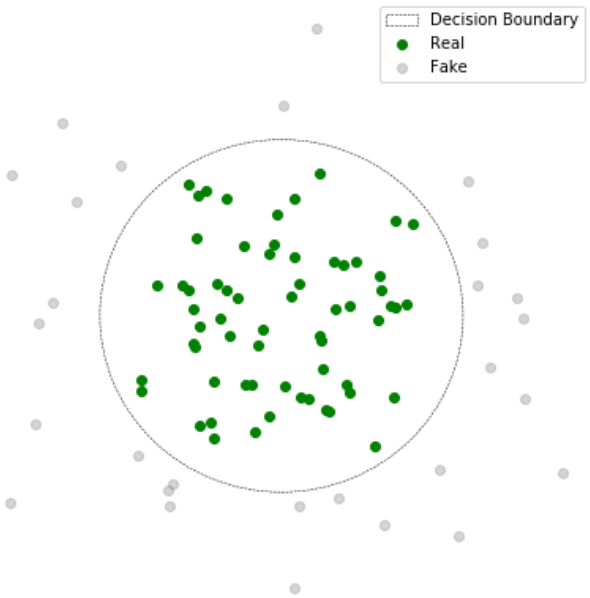


Figure 3.5: Visual representation of a good minimum using the AD4 method. Real examples are depicted as green dots and fake data as gray dots. The origin is depicted as '`x`' and the decision radius as a dashed line. The hypothesis justifying this method states that real data near the decision radius are more anomalous since they are closer to fake generated data than to the bulk of the real data.

Evaluation Of Approaches

Because there is usually no ground truth label specifying the level of anomalousness of the data, we will compare the two approaches instead by assessing how the two "anomaly scores" correlate when trained on the same dataset, and assess some of the high-scoring data points qualitatively to evaluate their anomalousness.

3.2.2 Supervised Classification

Supervised classification is a task wherein we are asked to predict which label(s) should be assigned to which data point, given a set of ground truth data-label associations. Using the training data, we should be able to train a model which accurately predicts labels for previously unseen data.

SC1

As before, we choose a method which is as simple as possible for the first approach. For this case, we are only interested in the classification performance of a classifier which predicts labels based on the data's representation at the code layer. Here we will use a vanilla neural network classifier at the code layer.

SC2

This method is exactly like SC1 except it uses the VQ classifier.

SC3

The final approach we evaluate for supervised classification is reminiscent of the second approach for unsupervised clustering. As before, we compute the pairwise distances for each data point's code representation between itself and all exemplars, then apply a kernel to the pairwise distances and then apply a softmax layer. This time our objective is to minimize the cross-entropy between the softmax predictions and the one-hot vector indicating the true label. See Figure 3.6 for a visual aid.

Evaluation Of Approaches

Evaluation of the supervised classification task proceeds like normal, namely computing some metric of accuracy (*e.g.*, precision, recall, or F1-score) of the predictions after training.

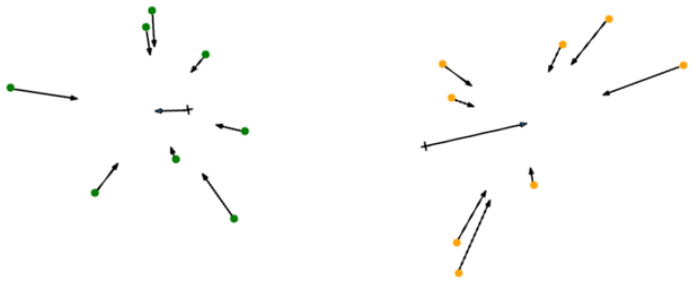


Figure 3.6: Visual representation of the gradients associated with an update using the SC3 method. Green dots and orange dots are examples belonging to two different classes. Each exemplar vector is represented as '`x`' and the gradients are shown as arrows.

4

Evaluation

We assess the performance of each method in Section 3.2 on a random subset of the MNIST handwritten digits dataset, where the data is flattened into 1D vectors, z-normalized, and ingested into the system. Then we train the system and compute metrics indicating the success of the system to learn to perform the associated task. We also evaluate the training process using task-specific means.

We define two architectures which differ primarily in the width of the code layer so that we can also evaluate the effect of the "information bottleneck" induced by it versus the autoencoder objective. All details of the experimental setup can be found in Appendix A, but for clarity and brevity we will mention here that the primary axes along which we evaluate the system are the hyperparameter associated with the autoencoder term in the objective (here denoted λ) and the width of the code layer.

Throughout this section, shorthand is used to refer to the various experiments. The format is simple: *AA-BB* where *AA* specifies the method used and *BB* indicates the width of the code layer.

High-Dimensional Visualization

For all figures for which we must visualize a high-dimensional space, we first apply nonlinear dimensionality reduction using the UMAP algorithm [13].

The Original MNIST Data Space

A dimensionality-reduced visualization of the MNIST dataset in its original space is found in Figure 4.1.

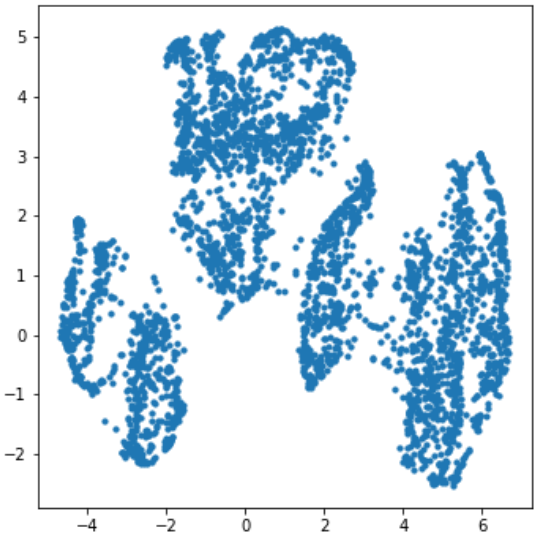


Figure 4.1: Distribution of the MNIST dataset in the original space, visualized using UMAP. Class label information is deliberately eschewed to emphasize the structure. Even without labels, the data exhibits clusters which are readily recognized.

4.1 Anomaly Detection

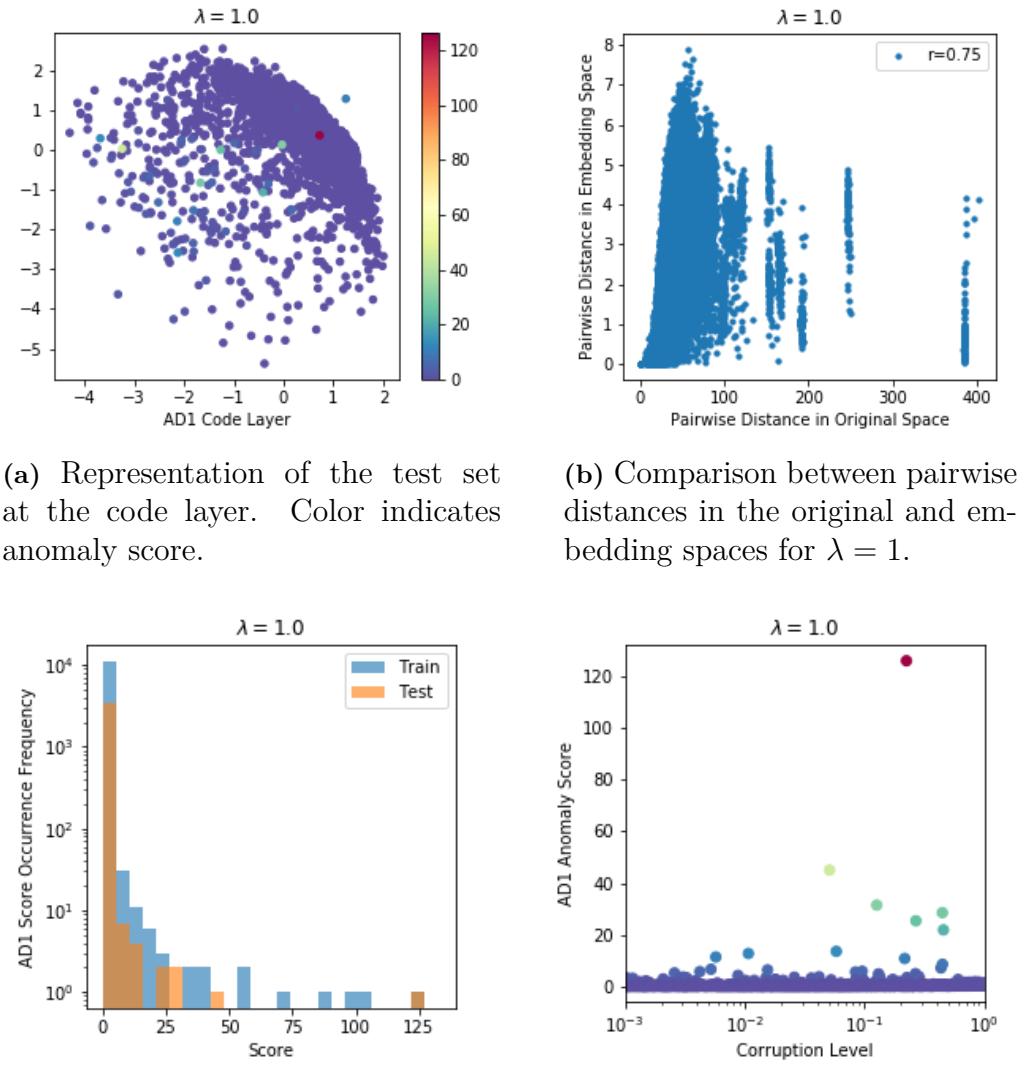
Data Corruption

The data is initially z-normalized, meaning the whole dataset undergoes an affine transformation so that the mean of each feature is zero and the variance of each feature is one. This puts a standard Gaussian prior probability distribution on the data, if we are considering it in a Bayesian context. To corrupt the data, we add Gaussian noise with a specific standard deviation. For these experiments we vary the magnitude of the Gaussian noise smoothly from 10^{-3} to 1, which corresponds to a signal-to-noise ratio (SNR) which ranges from 1000 to 1. In real terms, this means we corrupt each element of the test dataset with some value between this range and assess the system for its ability to reliably score data higher when it is more corrupted.

When assessing the below methods, we use the unedited test set for visualizing the code layer and scores, and only use the corrupted dataset to study the response to data corruption.

4.1.1 AD1-2

All values of λ will produce similar results for this method as the only term in the objective function is the autoencoder term. Different values for λ will only change the effective step size during training, which can be accounted for by correspondingly



(a) Representation of the test set at the code layer. Color indicates anomaly score.

(b) Comparison between pairwise distances in the original and embedding spaces for $\lambda = 1$.

(c) A histogram of score assignments for training and test sets.

(d) No response to corrupted data.

Figure 4.2: Characterizing the performance and representation at the code layer of the AD1 method with code layer width 2.

changing the learning rate. For these reasons we only test this method for $\lambda = 1$. The results are found in Figure 4.2.

To human eyes, the representation of the data at the code layer (see Figure 4.2a) seems to be surprisingly uninformative. The data which are poorly reconstructed are also in unexpected locations relative to the rest of the data, although perhaps that is not so surprising since the data may not "belong" in that location in the code layer in the first place.

The pairwise distances between data in both spaces, while relatively highly correlated, still exhibits unconventional patterns. Most notable is that there are distinct "echelons" in the distances in the original space indicated by the vertical stripes on the right side of the plot in Figure 4.2b.

The histogram of score assignments in Figure 4.2c is useful in this and the next analyses for inferring appropriate anomaly thresholds and investigating the generalization capacity of the method to unseen data. Insights gleaned from these histograms would inform the training process and interpretation of the scores. In the case of AD1, we see that a large majority of data points are given very low scores, indicating nominal behavior, and only a very few deviate from that distribution, in keeping with how we usually interpret anomalies as extreme events. The distribution of scores for the training set is surprisingly wider than that of the test set, which we would expect to receive higher scores on average since they are not guaranteed to be "in distribution".

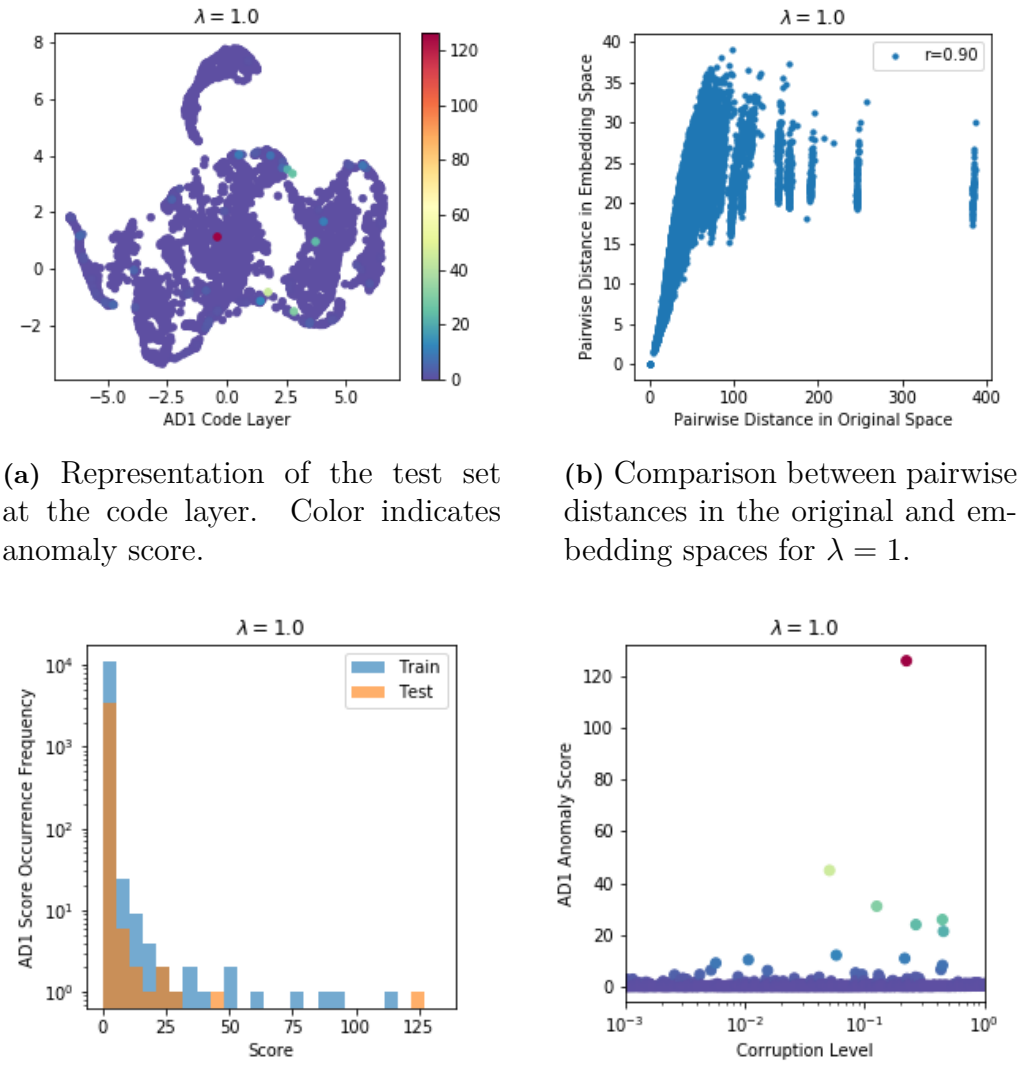
The assigned scores are almost completely independent of the corruption level, with a very slight positive trend for just a couple data points. It is possible that the points which are scored highly are true anomalies, not just because of the artificial corruption process, but without ground truth labels we cannot assess this objectively.

4.1.2 AD1-784

The results depicted in Figure 4.3 indicate that the performance of AD1 does not significantly change as code layer width is increased. The response to corrupted data has a very slight positive trend but does not indicate reliable anomaly detection. The biggest difference between the two conditions is that the pairwise distances are much more highly correlated, likely due to the fact that in the absence of any incentive to compress the data meaningfully, all the layers in the autoencoder will eventually learn to approximate the identity function, and the correlation of pairwise distances approaches 1.

4.1.3 AD2-2

For this method, we do not test $\lambda = 0$ because we would achieve a trivial solution where every data point is mapped to the origin in the code layer.



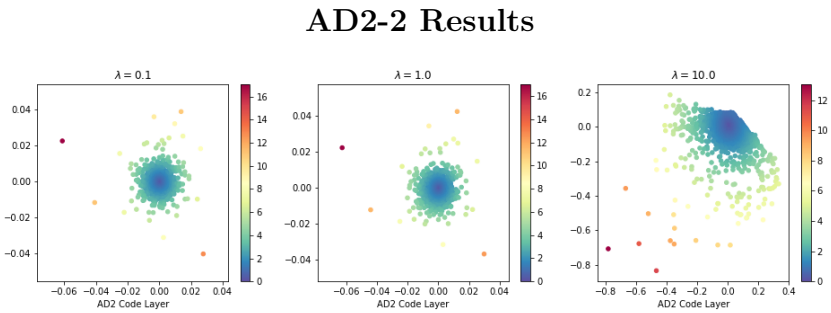
(a) Representation of the test set at the code layer. Color indicates anomaly score.

(b) Comparison between pairwise distances in the original and embedding spaces for $\lambda = 1$.

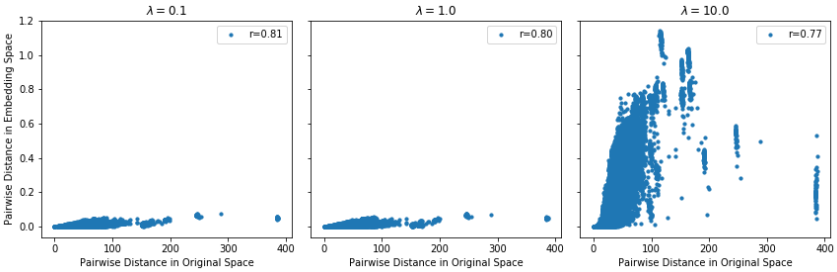
(c) A histogram of score assignments for training and test sets.

(d) No response to corrupted data.

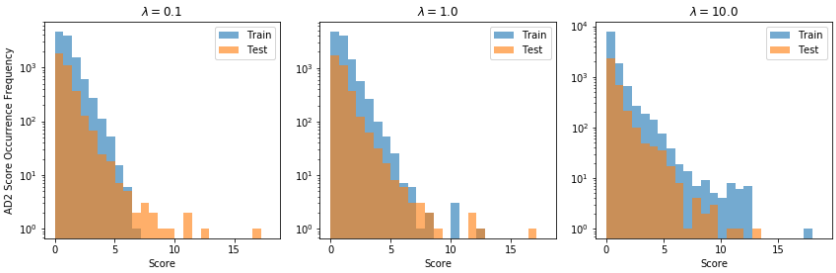
Figure 4.3: Characterizing the performance and representation at the code layer of the AD1 method with code layer width 784.



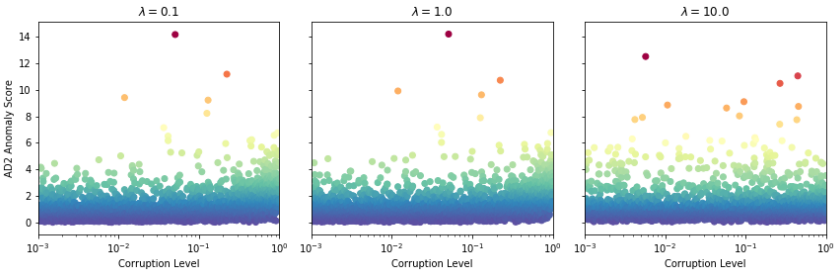
(a) Representation of the test set at the code layer. Color indicates anomaly score.



(b) Comparison between pairwise distances in the original and embedding spaces for varying λ.



(c) A histogram of score assignments for training and test sets.



(d) Slight response to corrupted data.

Figure 4.4: Characterizing the performance and representation at the code layer of the AD2 method with code layer width 2.

The code layer looks pretty much as expected, distributed roughly Gaussian or Laplacian about the mean for lower values of λ. The exception is when the autoencoder objective dominates at $\lambda = 10$, where the data starts to resemble the code layer from AD1 more.

One thing to notice is that the magnitudes of the scores change drastically as λ varies under AD2. This is because without sufficient pressure to distinguish between data points, they will collapse to very near the origin.

The histogram depicts a roughly exponential distribution of scores with no clear boundary between what may be considered nominal versus anomalous. It would be difficult to act on this information without labelled data to select a decent alarm threshold.

We can see a very slight response to the corrupted data for corruption level $> 20\%$, but there is not enough to assume that it will reliably detect known anomalies.

4.1.4 AD2-784

Compared to having a code layer width of 2, this method exhibits a strong response to corrupted data. However, there are also a number of false positives that disqualify this method from being considered for an efficient anomaly detection application.

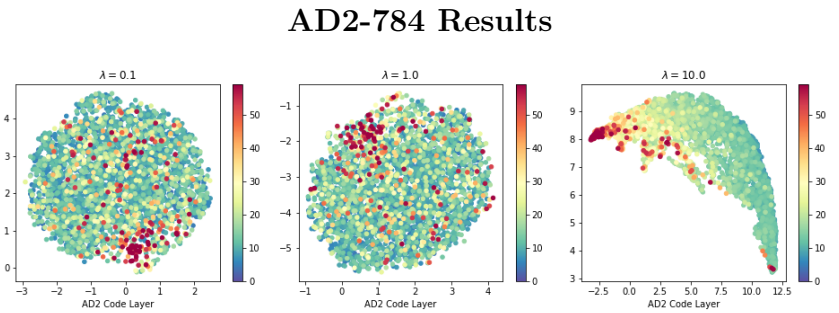
Typically, data which is out-of-distribution with respect to the training data *should* receive higher scores as it fits "worse" to the distribution learned by the model. However, in this case the test data received lower scores on average than the training data, defying the intuition about anomaly detection algorithms.

Recall that the training regimen specified a small number of training epochs, meaning that the model may not have converged to its stationary state. Increasing the number of training epochs may eventually remove most or all the false positives.

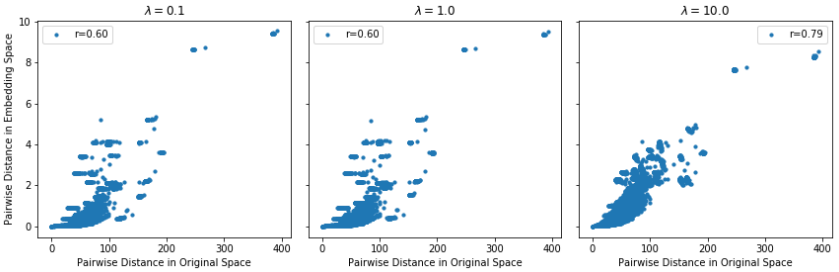
4.1.5 AD3-2

The differences between code layer representations for varying λ are surprisingly small. The exception is $\lambda = 10$ where something clearly went awry during training. The leading hypothesis is that during training, some "fake" data points were generated *very* near the origin, resulting in massive gradients. The update step then took a massive step and momentum carried the data points' representations away from the origin. When the momentum finally dissipated, there was not sufficient time in the training or strength of gradients to pull those points back toward the origin.

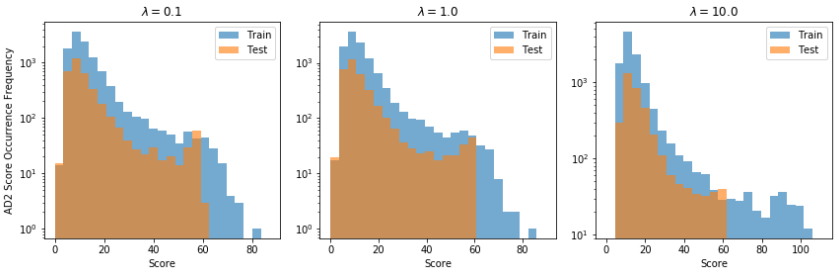
The score distribution takes on an interesting shape that evolves as λ varies. Below $\lambda = 1$, the distribution of scores is smooth and there seems to be one contiguous body of scores, meaning it is hard to set a threshold for the anomaly alarm. However, at $\lambda = 1$ and above, some of the data points separate from the main mass and two distinct clusters emerge, making threshold choice easier.



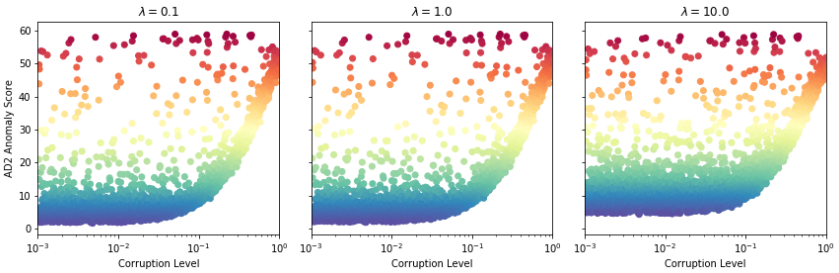
(a) Representation of the test set at the code layer. Color indicates anomaly score.



(b) Comparison between pairwise distances in the original and embedding spaces for varying λ.



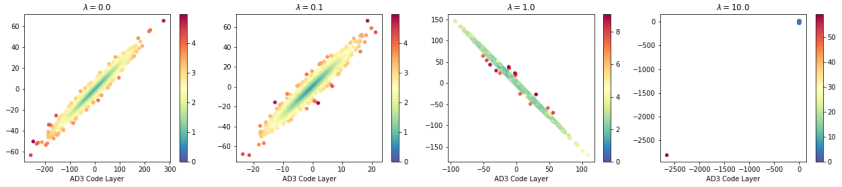
(c) A histogram of score assignments for training and test sets.



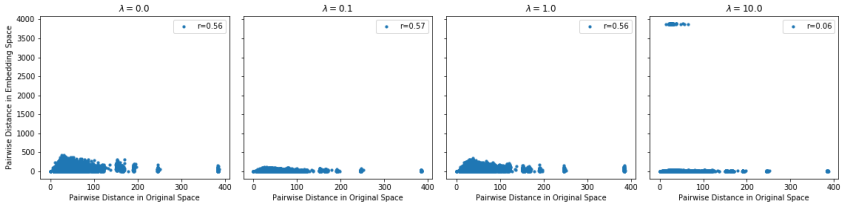
(d) Slight response to corrupted data.

Figure 4.5: Characterizing the performance and representation at the code layer of the AD2 method with code layer width 784.

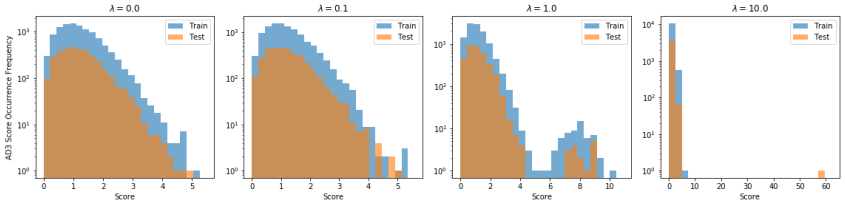
AD3-2 Results



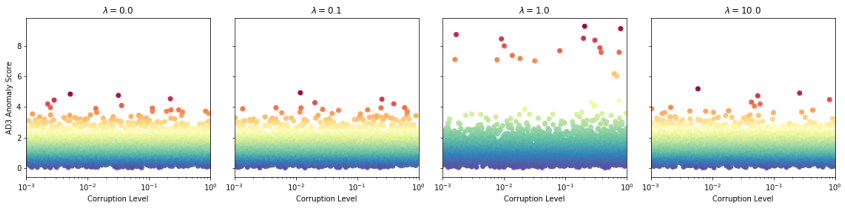
(a) Representation of the test set at the code layer. Color indicates anomaly score.



(b) Comparison between pairwise distances in the original and embedding spaces for varying λ.



(c) A histogram of score assignments for training and test sets.

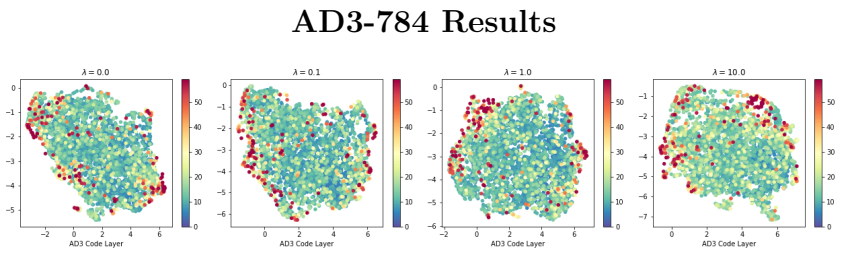


(d) Response to corrupted data.

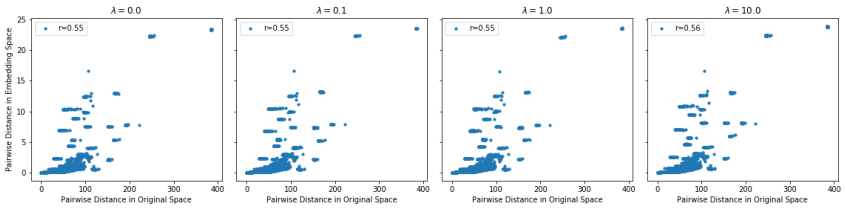
Figure 4.6: Characterizing the performance and representation at the code layer of the AD3 method with code layer width 2.

Unfortunately, there is no perceptible response to corrupted data.

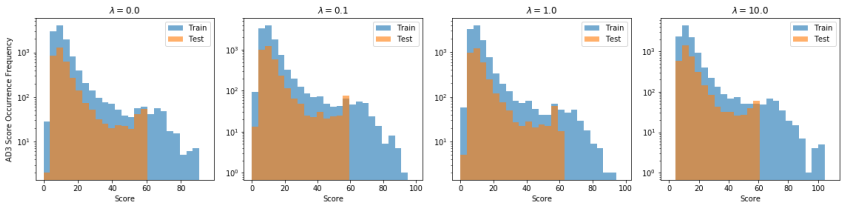
4.1.6 AD3-784



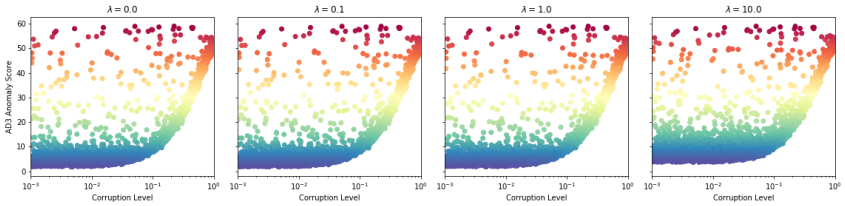
(a) Representation of the test set at the code layer. Color indicates anomaly score.



(b) Comparison between pairwise distances in the original and embedding spaces for varying λ.



(c) A histogram of score assignments for training and test sets.



(d) Response to corrupted data.

Figure 4.7: Characterizing the performance and representation at the code layer of the AD3 method with code layer width 784.

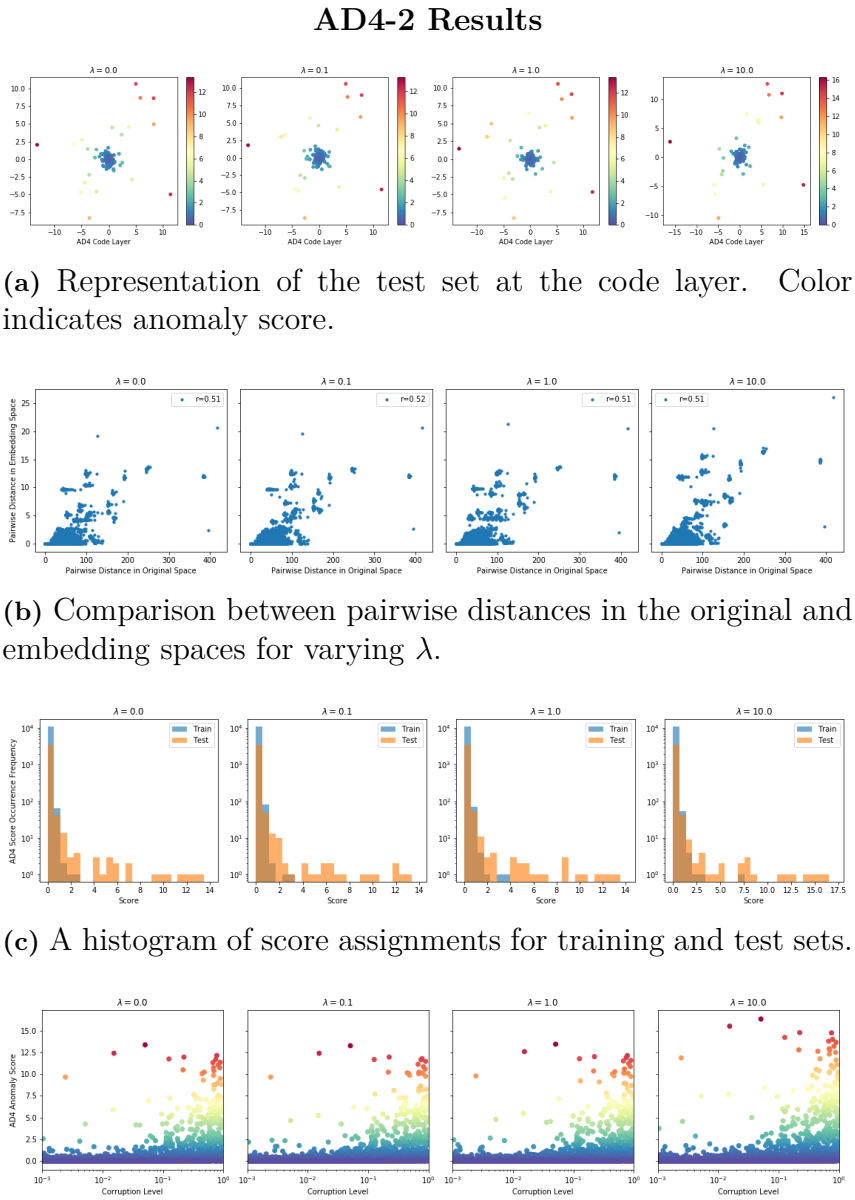
The differences between code layer widths 2 and 784 for this method reflect those for AD2. We see concerning behavior in that the test scores are lower than the training scores and the number of false positives remains high.

For this method, because we are generating fake data at every training iteration, there may not exist a proper stationary state, instead achieving some "quasi-stationary"

distribution which is regularly perturbed by fake data which is arbitrarily close to the distribution of the real data. So even though the results indicate the training process may look similar to that of AD2, it may be the case that training longer will not achieve the desired results. For this reason we consider AD3 to be worse than AD2.

Curiously, there is very little difference in results across varying λ.

4.1.7 AD4-2



(a) Representation of the test set at the code layer. Color indicates anomaly score.

(b) Comparison between pairwise distances in the original and embedding spaces for varying λ.

(c) A histogram of score assignments for training and test sets.

(d) Response to corrupted data.

Figure 4.8: Characterizing the performance and representation at the code layer of the AD4 method with code layer width 2.

Of all the methods studied in this thesis, this one is the most promising for practical use and represents the main novelty and contribution to the field.

The code layer is as we might expect, given the description of the method: a dense cluster near the origin with a smattering of points outside.

The pairwise distance comparison is also very informative, and indicates that this method is promising for anomaly detection as well as preserving relationships between data for *e.g.* nearest neighbor search. Note the distinct clusters of points visible in these comparisons: this indicates that relationships between points are upheld and the latent structure of the dataset is preserved to a high degree for these distinct clusters.

It is easy to interpret the score histograms as mixture distributions: one component distribution being very tightly distributed near zero, and the other distributed more uniformly over a much wider range, à la the 'spike-and-slab' model [14]. Furthermore, we can derive a principled first candidate for an anomaly threshold directly from the trained model by using the decision radius of the employed classifier, which provides additional interpretability of the resulting distribution of the data in the code layer.

We see a strong response to the artificial anomalies as well. Starting at a corruption level of about 10%, the assigned score increases with increasing corruption up to near the maximum assigned score among all points. This is a strong result and shows that, at least in the case of Gaussian noise, this detector is reasonably reliable.

4.1.8 AD4-784

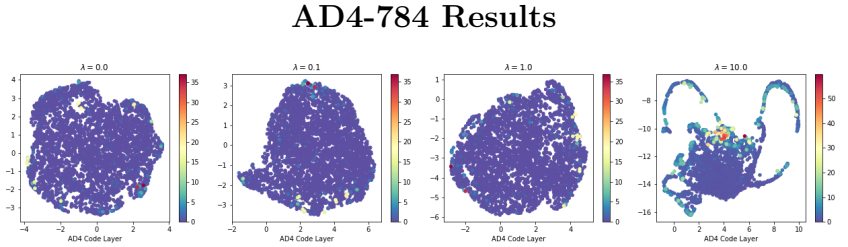
The differences between narrow and wide code layers for this method are minor, however there is a more consistent response to corrupted data. For large λ, the number of false positives increases significantly, and the score distribution histogram loses its beneficial "spike-and-slab" structure so that determining the alarm threshold is not as straightforward.

4.2 Supervised Classification

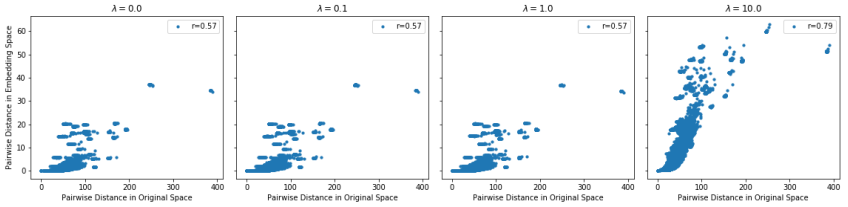
4.2.1 SC1-2

As λ varies, the representation of the data at the code layer undergoes only minor changes. As λ grows, the organization of the code layer becomes more "noisy" with respect to the classification task and the clusters become more spread out.

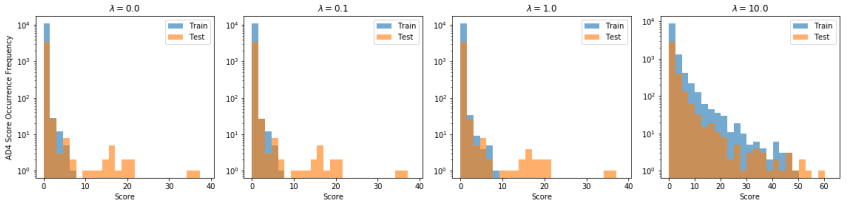
The training loss acts approximately as expected for such a training task. However, the validation loss exhibits interesting behavior: the validation loss first drops very quickly and then rises again, with a sharp transition between the two behaviors. This



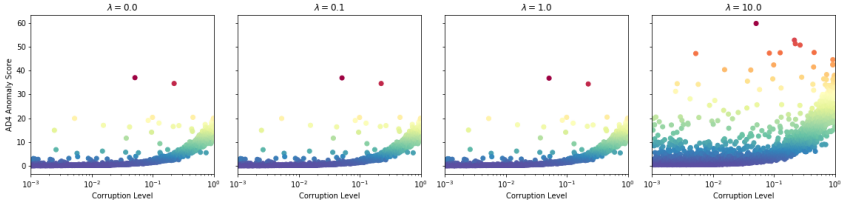
(a) Representation of the test set at the code layer. Color indicates anomaly score.



(b) Comparison between pairwise distances in the original and embedding spaces for varying λ.

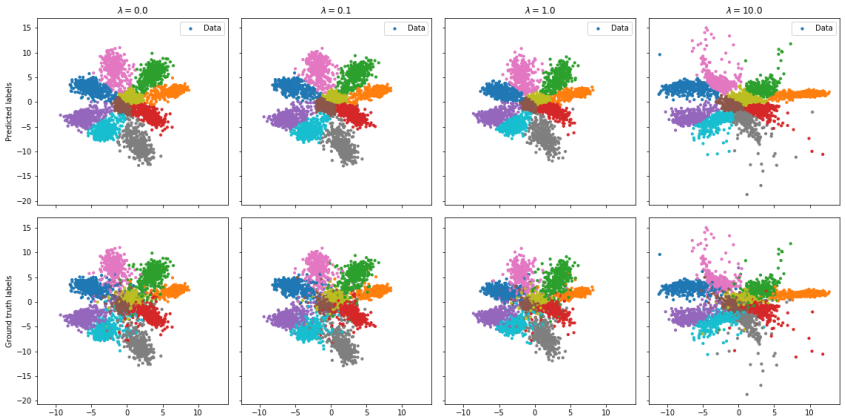


(c) A histogram of score assignments for training and test sets.

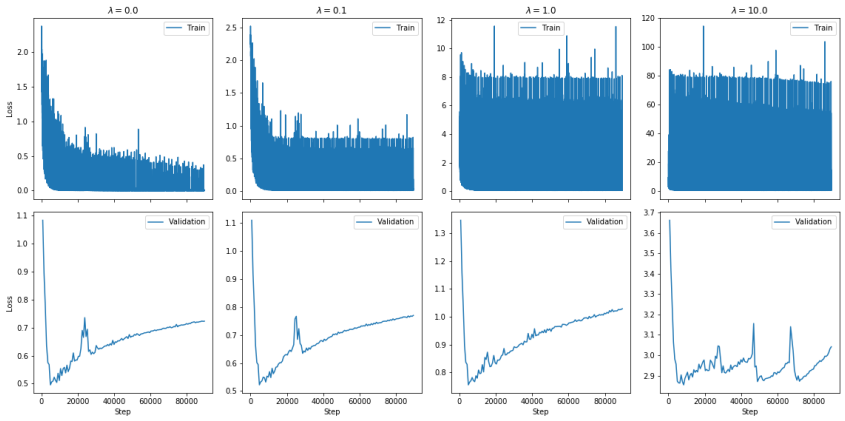


(d) Response to corrupted data.

Figure 4.9: Characterizing the performance and representation at the code layer of the AD4 method with code layer width 784.



(a) Representation of the test set at the code layer. Color indicates class labels: predicted label on the top row and ground truth label on the bottom row.

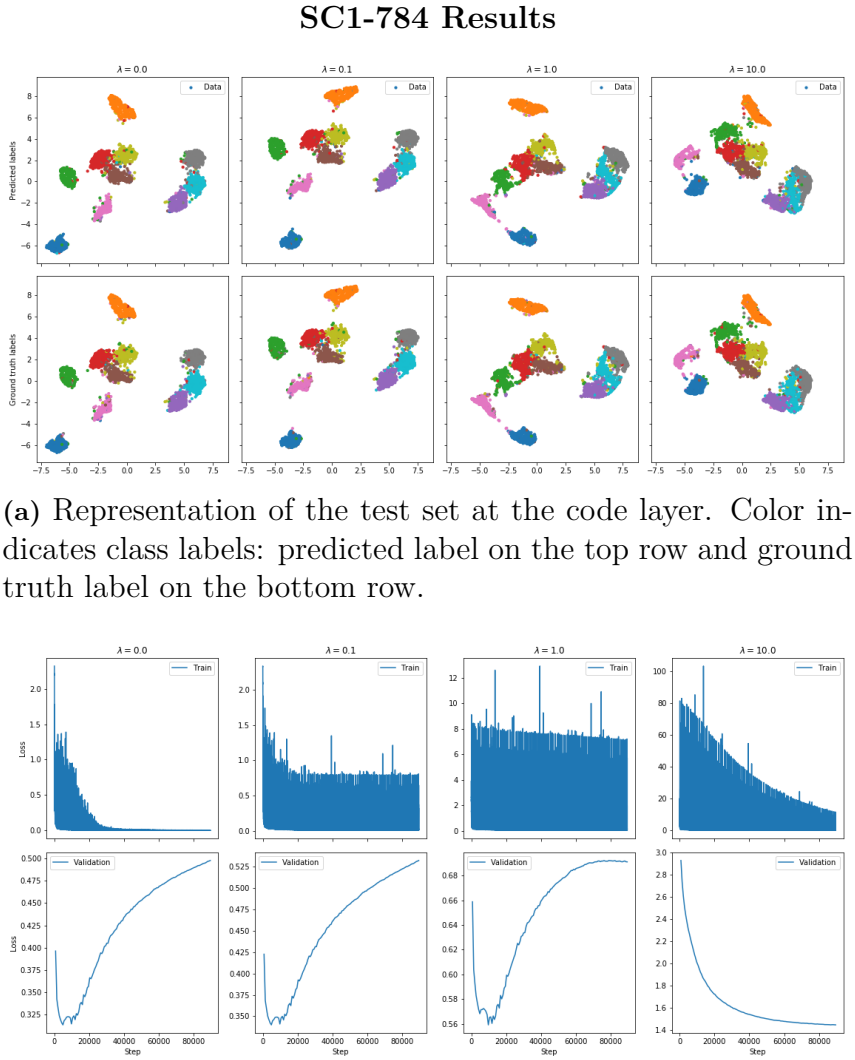


(b) Training and validation loss histories over the course of training.

Figure 4.10: Characterizing the performance and representation at the code layer of the SC1 method with code layer width 2.

is true for most values of λ, however as it grows to near $\lambda = 10$ the validation loss shows more erratic behavior. In this case we see that the model has not yet reached a stationary state and continues to "bounce" as the autoencoding and classification objectives compete. Further tests are necessary to determine the long-term behavior.

4.2.2 SC1-784



(a) Representation of the test set at the code layer. Color indicates class labels: predicted label on the top row and ground truth label on the bottom row.

(b) Training and validation loss histories over the course of training.

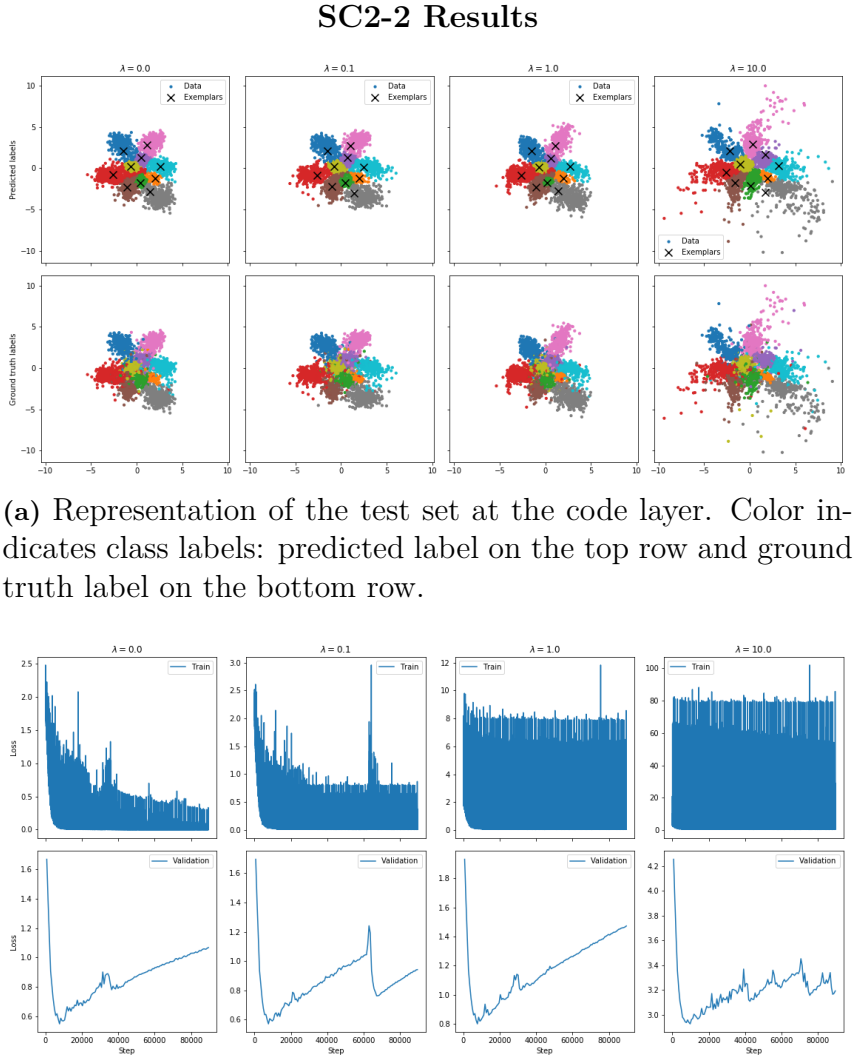
Figure 4.11: Characterizing the performance and representation at the code layer of the SC1 method with code layer width 784.

In contrast to having a code layer of width 2, the wide code layer allows for much more separation between labelled clusters. As indicated by the loss curve for $\lambda = 0$, training loss approaches zero and the clusters are properly labelled. However, the validation curve very quickly transitions from decreasing to increasing, and does so before the model converges with respect to the training data. This may indicate that

more training epochs are necessary to achieve a true convergence to a generalizable classification model.

The exception again is $\lambda = 10$, where the losses only decrease. Presumably this is because the autoencoder objective has not yet reached a minimum and is still optimizing after the training epochs.

4.2.3 SC2-2



(a) Representation of the test set at the code layer. Color indicates class labels: predicted label on the top row and ground truth label on the bottom row.

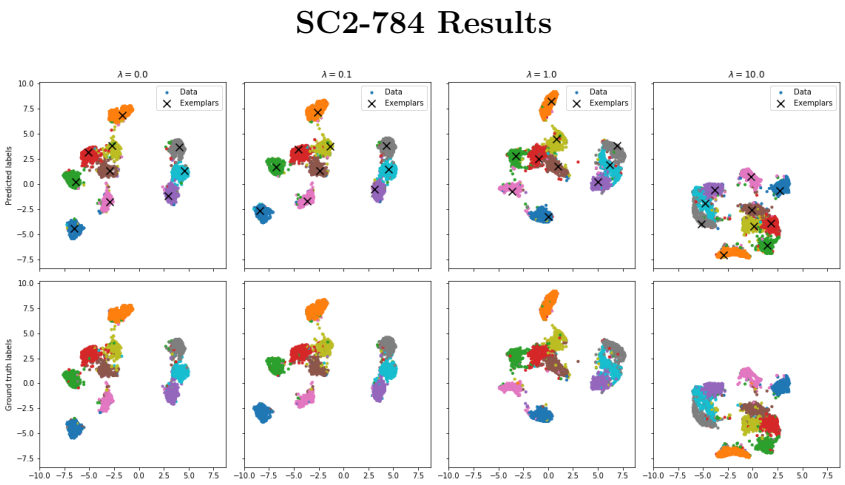
(b) Training and validation loss histories over the course of training.

Figure 4.12: Characterizing the performance and representation at the code layer of the SC2 method with code layer width 2.

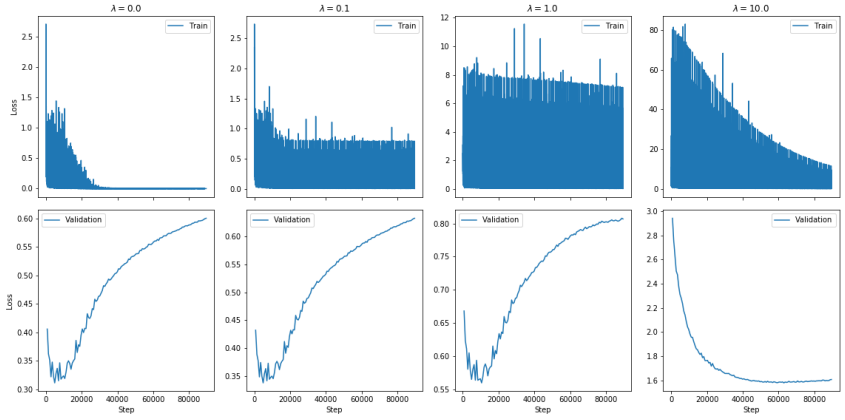
The autoencoder objective exhibits the same behavior as before, spreading out the data in the code layer. In general, though, the conclusions made about SC1 also apply here. Clearly the training will take a long time to converge, but it is less

clear how the performance will change as the autoencoder objective in particular converges.

4.2.4 SC2-784



(a) Representation of the test set at the code layer. Color indicates class labels: predicted label on the top row and ground truth label on the bottom row.

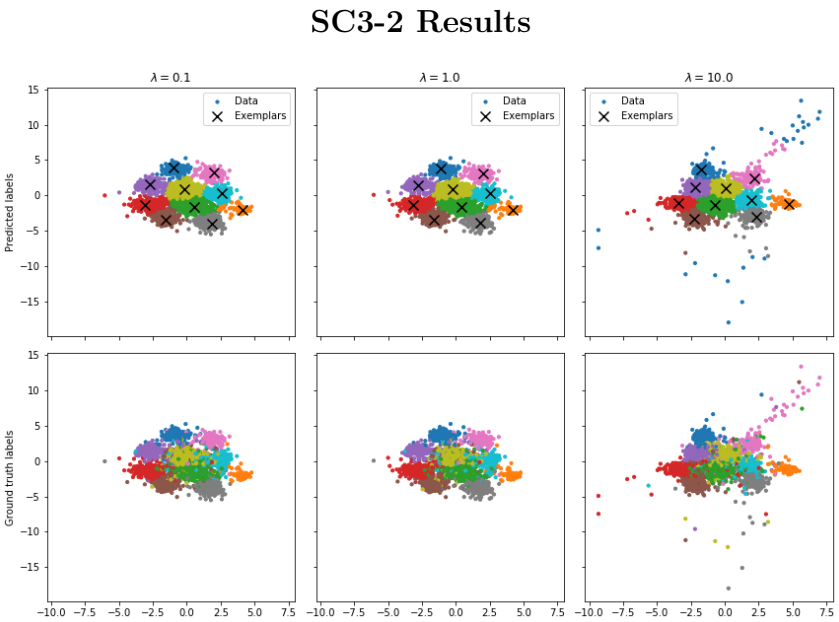


(b) Training and validation loss histories over the course of training.

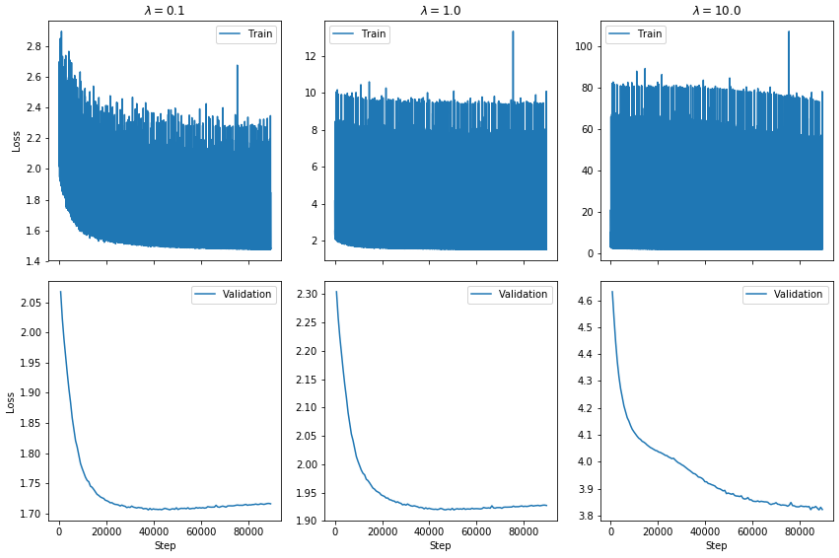
Figure 4.13: Characterizing the performance and representation at the code layer of the SC2 method with code layer width 784.

As before, the clusters become less distinct as λ grows. The conclusions that can be drawn from these results are tenuous, but they may indicate that in the absence of an autoencoder objective, the system already has sufficient capacity to meaningfully separate the clusters.

4.2.5 SC3-2



(a) Representation of the test set at the code layer. Color indicates class labels: predicted label on the top row and ground truth label on the bottom row.



(b) Training and validation loss histories over the course of training.

Figure 4.14: Characterizing the performance and representation at the code layer of the SC3 method with code layer width 2.

The clusters here are much closer together and more mixed with each other than previous methods. This indicates that the gradients associated with the form of loss function used for these experiments are not very strong compared to the other

methods. The representation at the code layer for the $\lambda = 10$ case is to be expected based on our experience with the previous methods.

We also see more typical validation loss curves, which drop quickly (initial training) before plateauing (quasi-convergence) and beginning to rise slowly (overfitting). However, the training loss curves indicate that training to convergence will take a long time and so this method may be unsuitable for larger models and datasets due to computational resource and time restrictions.

4.2.6 SC3-784

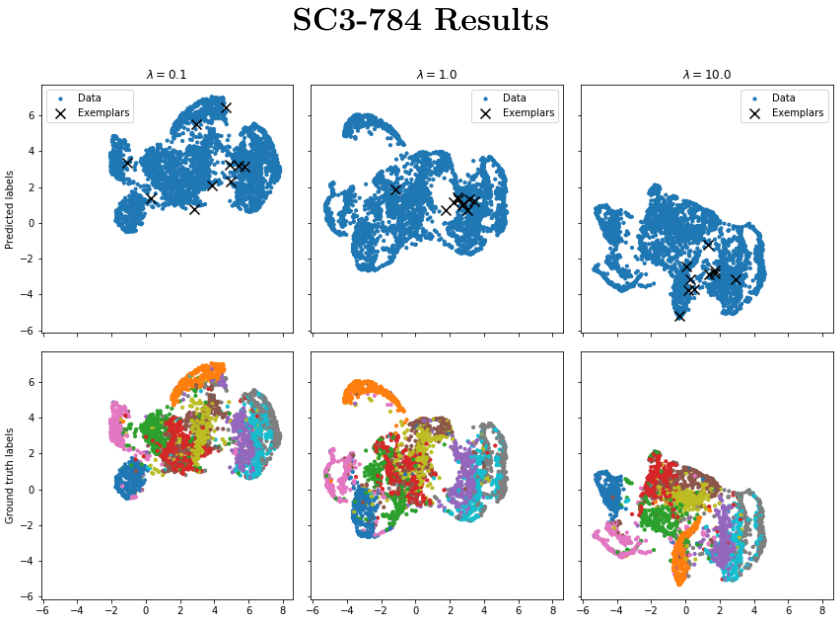
The results of this experiment indicate once again that the gradients are not so strong for this method as compared to other methods, seen here by inspecting the representation at the code layer and noting the clusters corresponding to class labels are still mixed with each other. This reinforces the conclusion that this method does not train so quickly compared to others and more epochs will be necessary to converge. This is in general a negative result because we would like models which converge quickly to good performance on the related task(s).

4.2.7 Quantitative Results

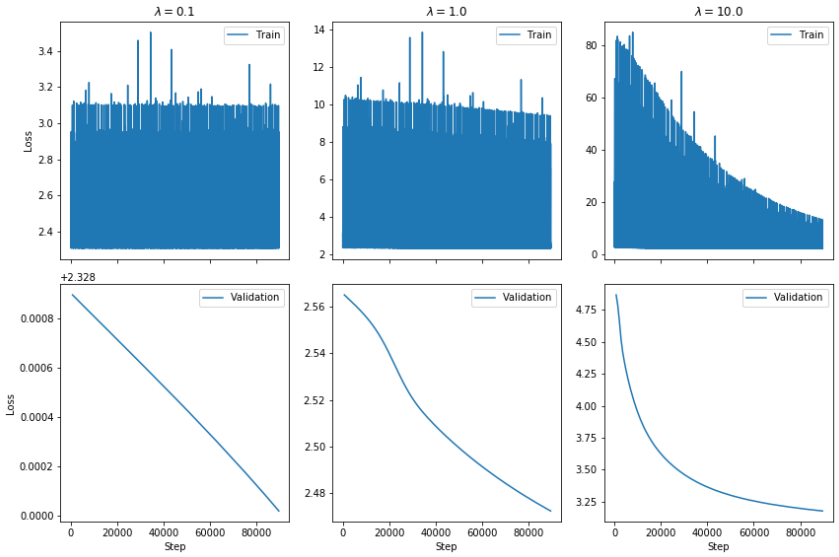
Task	Code Layer Width	$\lambda = 0.00$	$\lambda = 0.10$	$\lambda = 1.00$	$\lambda = 10.0$
SC1	2	0.897	0.900	0.889	0.890
	784	**0.939**	**0.939**	0.938	0.931
SC2	2	0.889	0.891	0.875	0.884
	784	**0.937**	**0.937**	0.934	0.922
SC3	2	∅	**0.881**	**0.881**	0.859
	784	∅	0.098	0.098	0.098

Table 4.1: Accuracy results for all supervised classification experiments. Best performance per task is highlighted in bold.

Table 4.1 shows the accuracy results for every experiment performed on the MNIST dataset. The entries for SC3 with $\lambda = 0$ are omitted because the minimum of the loss function is a trivial representation, so these experiments were not performed.



(a) Representation of the test set at the code layer. Color indicates class labels: predicted label on the top row and ground truth label on the bottom row.



(b) Training and validation loss histories over the course of training.

Figure 4.15: Characterizing the performance and representation at the code layer of the SC3 method with code layer width 784.

5

Conclusion

The framework presented in this thesis has the stated objective of improving performance on typical tasks performed by neural networks using an augmented objective function. This objective function includes an additional term corresponding to reconstruction of the input given the representation of the data at some intermediate bottleneck layer in the network, referred to in this document as the "autoencoder objective".

For the purposes of anomaly detection, four different methods were explored. These methods differ in complexity and the gradients associated with the training procedure, and each also invites different interpretations of the resulting representation for the purposes of anomaly scoring and raising alarms. Of all the methods, AD4 was the most promising for detecting true out-of-distribution examples and scored examples proportionally to the level of added noise in the data.

For supervised classification, three methods were explored and evaluated according to the usual metrics. The best was the simplest method, SC1, which uses only a linear classifier at the code layer of the autoencoder.

We find that, across all tasks, when the code layer width is too restricted, the performance is generally worse. That is, the code layer acts as a bottleneck whose impact on the performance dominates over the autoencoder objective. In the case of anomaly detection, this corresponds to more false positives and negatives, and for classification, this means lower classification accuracy. The primary issue here is that of insufficient flexibility in the code space to organize the data efficiently for the associated task—a full-width code layer is most likely excessive, and a balance can be achieved in code layer width which is optimal in this context, compromising between code space flexibility and model complexity.

We also find that increasing the influence of the autoencoder objective tends to decrease the task performance, presumably because the gradients associated with the autoencoder objective are too strong and bring the model away from configurations which result in good performance. This assessment should only be restricted to the architectures used in these experiments, mainly because the networks were defined such that both the encoder and decoder halves have sufficient "capacity" to efficiently separate the classes by the time the data reaches the code layer. Perhaps for networks which are too small (too narrow or too shallow), the autoencoder objective may improve the performance, but we may also see that the upper bound on accuracy is

lower for such small networks.

All experiments were performed with a minimum of training time for expediency and to "stress-test" the various methods in this thesis. The motivation for this is to evaluate how quickly each method reaches a reasonable solution, not necessarily to produce the networks with the best performance overall. In future experiments, it would be fruitful to explore the most promising methods and architectures and train them to convergence in case the long-term behavior diverges from what is observed in earlier stages of training.

Future work on the framework introduced in this thesis could explore: the effect of different activation functions such as ReLU on the performance of the various methods; extensions to other network architectures such as CNNs and RNNs; use of more sophisticated "fake data" generation methods such as GANs or Gaussian processes for the anomaly detection methods which require them; further inspection of the structure of the code space, in particular the results of decoding learned exemplar vectors to understand what they represent; or more details on integration into existing data analysis pipeline.

Bibliography

[1] R. Bellman, Rand Corporation, and Karreman Mathematics Research Collection. *Dynamic Programming*. Rand Corporation research study. Princeton University Press, 1957.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[3] Y. Bengio, A. Courville, and P. Vincent. Representation Learning: A Review and New Perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, aug 2013.

[4] H. N. Mhaskar and C. A. Micchelli. How to choose an activation function, 1993.

[5] Normand J. Beaudry and Renato Renner. An intuitive proof of the data processing inequality. jul 2011.

[6] Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. oct 2016.

[7] R. Linsker. Self-organization in a perceptual network. *Computer*, 21(3):105–117, mar 1988.

[8] Naftali Tishby and Noga Zaslavsky. Deep learning and the information bottleneck principle. In *2015 IEEE Information Theory Workshop (ITW)*, pages 1–5. IEEE, apr 2015.

[9] Rich Caruana. Multitask Learning. *Machine Learning*, 28(1):41–75, 1997.

[10] Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. Convex multi-task feature learning. *Machine Learning*, 73(3):243–272, dec 2008.

[11] Minmin Chen, Kilian Q. Weinberger, and John C. Blitzer. *Co-training for domain adaptation*. Neural Information Processing Systems, 2012.

[12] Bernardino Romera Paredes, Andreas Argyriou, Nadia Berthouze, and Massimiliano Pontil. Exploiting Unrelated Tasks in Multi-Task Learning, mar 2012.

[13] Leland McInnes, John Healy, and James Melville. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. feb 2018.

[14] T. J. Mitchell and J. J. Beauchamp. Bayesian Variable Selection in Linear

Regression. *Journal of the American Statistical Association*, 83(404):1023–1032, dec 1988.

[15] GradientBased Learning Applied to Document Recognition. In *Intelligent Signal Processing*. IEEE, 2009.

A

Implementation

All implementation and experiments were done using Python 3.5.2. Besides the built-in libraries, development proceeded with the following libraries:

- NumPy
- Pandas
- Matplotlib
- PyTorch

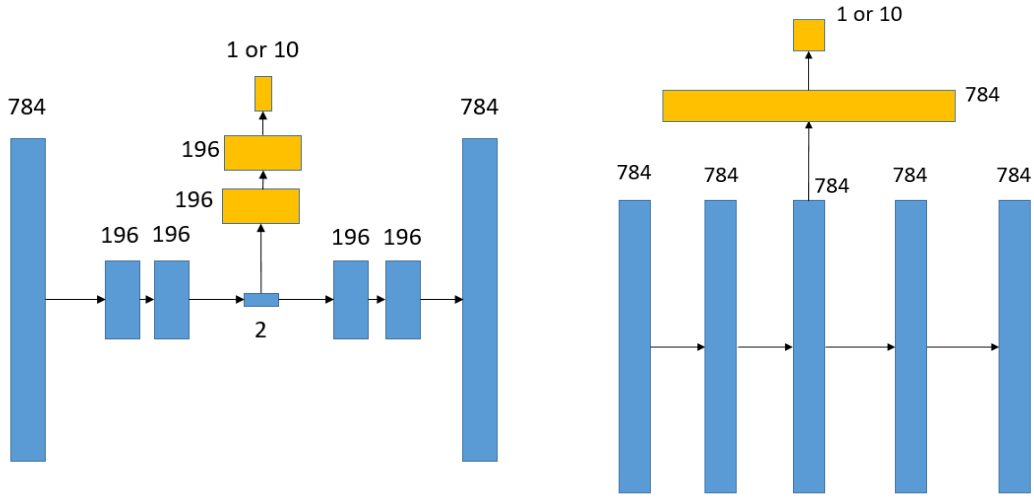
All of the following experiments were performed with the following optimization hyperparameters:

- Learning rate: $\eta = 1 \times 10^{-3}$
- Momentum: $\alpha = 0.9$
- Batch size: $B = 16$
- Epochs: $N_E = 128$

The dataset used is the MNIST handwritten digits dataset [15], which contains images of handwritten digits at a resolution of 28×28. For the purposes of this thesis, each example was flattened into a 1D vector of length 784.

Two architectures are tested: the first is intended to test the performance of an extremely-bottlenecked architecture, and the other to test the performance of a full-width autoencoder. Each architecture is depicted in Figure A.1. For testing the VQ classifier, the specified architecture for the classifier is entirely replaced by the VQ classifier.

The regularization hyperparameter values tested are $\lambda \in \{0, 0.1, 1, 10\}$. We consider these the most important variables to test as they represent respectively no, mild, regular, and extreme regularization.



(a) Extreme-bottleneck architecture. **(b)** Full-width architecture.

Figure A.1: The two architectures employed for these experiments. Blue boxes represent the layers relevant to the autoencoder, and yellow boxes represent the classifier being used at the code layer of the autoencoder. There are either 1 or 10 outputs on the classifier depending on whether the task is anomaly detection or supervised classification, respectively.