# Deep Semi-Supervised Embeddings for Dynamic Targeted Anomaly Detection

**Edmunds, Riley**
Bletchley Park Team, I.A.T.
riley_edmunds@intuit.com

**Feinstein, Efraim**
Bletchley Park Team, I.A.T.
efraim@intuit.com

**Abstract**

Dynamic targeted anomaly detection classification problems can fool traditional machine learning models. While it is relatively simple to delineate between average members of the majority class and the target class, it is tremendously difficult to differentiate between anomalous members of the majority class and the target class. We propose a neural network architecture that is able to tackle this problem in its entirety, by acknowledging and attending to each level of the classification hierarchy. We propose chaining a deep semi-supervised autoencoder with a supervised neural network, whose inputs are filtered latent space representations generated by the semi-supervised encoder. Our model is able to intelligently delineate both between conventional members of the majority class and the target class, and further, between anomalous members of the majority class and the target class. By tackling these interdependent hierarchically, and with deep architectures, we boast both greater control over the classification task and richer model expressiveness overall. Finally, we demonstrate the modularity of the proposed architecture, as allowing for parametrization of each component with LSTMs and CNNs.

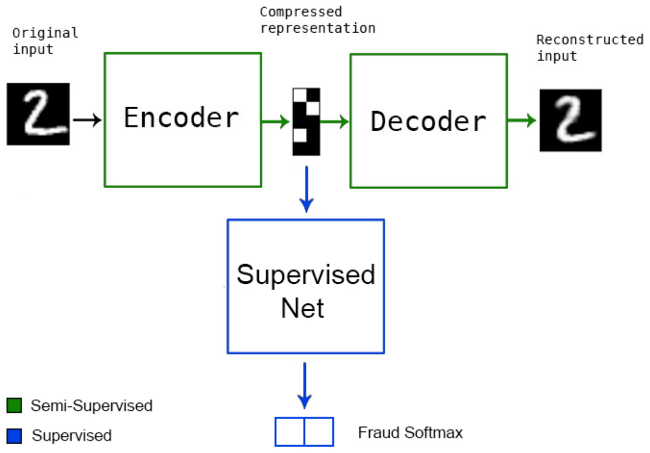

Figure 1: *Deep Semi-Supervised Embeddings Model*: the encoder and decoder are trained in a semi-supervised manner, and the supervised network is trained with supervision from both classes. Note that the encoder and decoder are trained until convergence before training the supervised network (See algorithm 1).

## Introduction

Consider small business owner John Smith's behavior today when registering an account on Mint. When first registering, Mint prompts new users to connect their credit cards. John has several, but is forgetful, and thus when connecting them to his account, mistypes his card numbers multiple times. To the human eye, while John is forgetful, he is certainly not committing fraud. However, Mint's fraud detection system is trained to detect "list validation attacks": credit card fraud attacks wherein the attacker obtains a list of credit card numbers illegally, and validates their legitimacy by connecting them in sequence to a Mint account – an attack oftentimes resulting in many authentication failures. Johns behavior resembles this attacks so closely that the fraud detection system would likely classify his behavior as fraudulent, and thus lock him out of his newly opened account. It is in Mint's best interest to recognize that John's behavior is not fraudulent – this could lead to a poor customer experience, and even to loss of a frustrated customer.

Keep in mind that it is relatively straightforward for a fraud detection system to recognize that a user who connects a single credit card without error is not committing a list validation attack. However, the true difficulty lies in distinguishing between Johns behavior (belongs to the **majority class** of non-fraudulent behavior, yet is an anomaly), and fraudulent user behavior, such as a list validation attack (the **target class** a fraud detection system would seek to identify). We address this problem in particular.

## Dynamic Targeted Anomaly Detection Problem

We formally define the generic class of problems resembling John's case as " dynamic targeted anomaly detection" problems – classification problems defined by the following constraints:

1. ***Skewed class sizes***
2. ***Concept drift in the target class***

3. ***Anomalies in the majority class***

The first two constraints motivate the use of unsupervised or semi-supervised learning. The final constraint (the existence of anomalies in the majority class) motivates the use of supervised learning.

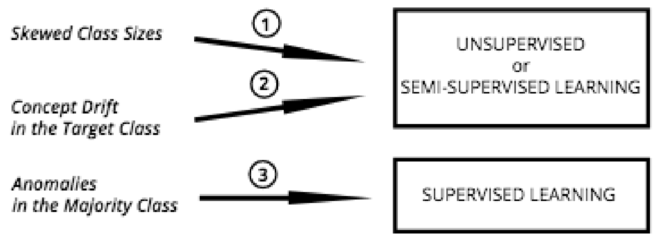
Skewed Class Sizes
1
Concept Drift in the Target Class
2
Anomalies in the Majority Class
3
UNSUPERVISED or SEMI-SUPERVISED LEARNING
SUPERVISED LEARNING

Figure 2: The "Dynamic Targeted Anomaly Detection" problem constraints and the corresponding learning algorithms best suited to address each constraint.

1. ***Skewed class sizes***: due to class size imbalance (majority class contains far more samples than the target class), we do not have sufficient target data to model the target class effectively in a supervised manner.
2. ***Concept drift***: the non-stationary data distribution of the target class motivates modeling solely the majority class (semi-supervised), or no classes at all (unsupervised). Even if we had enough target class data to model the target class effectively, we we would soon be modeling outdated data distributions, as the target class distribution is rapidly changing over time, as new data is collected (think of changing customer trends in the target class).
3. ***Anomalies in the majority class***: distinguishing between anomalous majority class user behavior (behavior which is formally labeled as belonging to the majority class, yet looks like data from outside this class) and target class user behavior is difficult because the anomalous majority class and target class have overlapping data distributions. Thus, we motivate the use of supervised methods to learn an effective decision boundary between these distributions.

To satisfy these three contradictory constraints simultaneously, we propose a hierarchical approach: first tackling the simple task of distinguishing between the expected members of the majority class and the target class, via deep semi-supervised learning, and finally distinguishing between the anomalous members of the majority class and the target class, via deep supervised learning.

Our contributions are as follows:

- We introduce a semi-supervised autoencoder as an anomaly detection algorithm, tailored to solve constraints inherent to the dynamic targeted anomaly detection problem domain.
- We extend this work to encompass the space of temporally dependent dynamic targeted anomaly detection problems.
- We extend these works to develop a final deep Semi-Supervised Embeddings model, which chains semi-supervised learning and supervised classification, in order to sole the Dynamic Targeted Anomaly Detection problem.

## Related Work

Previous work on semi-supervised autoencoders includes an autoencoder-based outlier detection method on MNIST (Lyudchik 2016). Chen et al. expands on this single-autoencoder model to train an ensemble of outlier-detection autoencoders (Chen et al. 2017).

Recent work in temporal deep representation learning has shown great success. Previous work on recurrent autoencoders includes the RNN Encoder-decoder paradigm, introduced by Cho et al. in (Sutskever, Vinyals, and Le 2014) and (Cho et al. 2014), which demonstrates the ability to embed temporal sequences in latent space, and reconstruct sequences with similar semantic structure from these latent embeddings. Google uses an extension upon this model (Wu et al. 2016) as their state-of-the-art translation engine. Li et al (Li, Luong, and Jurafsky 2015) explores sequence reconstruction with LSTM encoder-decoder networks. However, to our knowledge, there has been little to no published work in applying these deep representation learning techniques to the Dynamic Targeted Anomaly Detection problem.

Previous related work in chaining deep model architectures is limited to the following. Greedy layer-wise pre-training (Bengio et al. 2007) allows for learning neural network weights (in a supervised or unsupervised manner) in order to avoid the vanishing or exploding gradient problems. Dai et al (Dai and Le 2015) use semi-supervised recurrent neural networks to pre-train supervised networks. Transfer learning (Pan and Yang 2010) takes data representations learned on one task and uses them as a basis for learning a new task. This can consist of learning embeddings, which are then used to classify on a new task. However, these embeddings are typically learned both in a supervised manner, and with the scope of transferring representations from a learnable task, to one with little labeled data. For example, Ganin et al. (Ganin and Lempitsky 2015) propose a model able to take representations learned in a supervised manner, and transfer them to an unsupervised model. The architecture which most closely resembles ours is that of (Weston et al. 2012), wherein semi-supervised embeddings and supervised networks are trained jointly, but not in sequence, and without chaining the two architectures. To our knowledge, there do not exist any models which propose chaining semi supervised embedding with supervised classification.

## Theory

To best illustrate the model architecture and functionality, we first explain semi-supervised autoencoders, then extend them to a recurrent parametrization. Finally, we explain the Deep Semi-Supervised Embeddings model in full.

## Autoencoders

Autoencoders are neural networks characterized by lower-dimensional representations at some intermediate layer i. We can think of layer i as a sort of bottleneck, through which tensors must squeeze as they flow through the network. In place of traditional labels, autoencoders use ground truth input as supervision, and optimize for exact reconstruction of the input vector. Overall, we can think of autoencoders as learning an optimal nonlinear compression and reconstruction scheme under dimensionality-reduction constraints. Another interpretation of autoencoders is as a form of non-linear PCA, where the k principal components are analogous to the activations at layer i – the latent-space representation of the data. More precisely, for an input vector $x$, output reconstruction vector $\hat{x}$, and model weights $w$, we seek to solve the following optimization problem:

$$\arg\min_{w} L(x, z) = \frac{1}{n}\sum_{k=1}^{n} ||x - \hat{x}||_2^2$$

## Static Semi-supervised Autoencoder Model

We address anomaly detection in datasets plagued by skewed class sizes and concept drift by presenting a semi-supervised autoencoder, trained to reconstruct majority-class feature vectors by minimizing reconstruction error. Our architecture can effectively model any stationary DTAD problem. This architecture (1) Is robust to skewed class sizes, (2) Is robust to concept drift in the target class, (3) Offers superior and tunable expressiveness, (4) Does not require manual labeling of target-class samples.

At training time, we present the autoencoder with feature vectors representing the set of all majority-class data. We pass these into the model, and seek to minimize the reconstruction error between the output vectors and the ground truth feature vectors for each sample in the set of majority-class feature vectors. The autoencoder is forced to learn the optimal mapping to the latent space, and back to the original space, thus yielding the most ”information rich” latent space representation of majority-class feature vectors. In this manner, we intuit that the autoencoder weights learn mappings that best leverage the distribution of feature values belonging to majority-class transactions. We believe the compositional structure of the autoencoder network will allow for hierarchical and fundamentally more complex modeling of majority-class data.

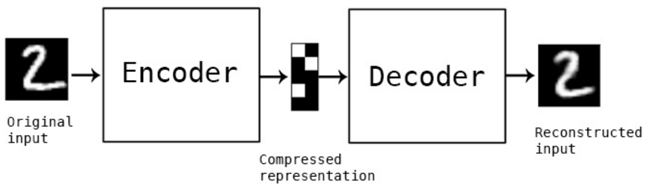

Figure 3: Encoder-Decoder network

At test time, we feed a list of unseen static feature vectors to the model. For each, we receive back an output vector in the same space as the input vector. We then compute the reconstruction error between each sample's original distribution feature value distribution and its output features value distribution. Herein lies the intuition: the model has been optimized to effectively reconstruct majority-class samples. Thus, we expect the reconstruction error for unseen majority-class samples to be generally smaller than the reconstruction error for unseen target-class examples. In other words, the autoencoder has learned the distribution of majority-class examples, and distance from this distribution can be quantified as reconstruction error. Thus, never-before-seen samples, whose feature value distribution differs greatly from the feature value distribution of majority-class samples, have high reconstruction error (think of normalized reconstruction error as a proxy for probability that the given sample belongs to the target class). We post-process the model outputs by ranking the corresponding samples in order of decreasing reconstruction error. We then encourage manual inspection of the top n samples in this set, where n a parameter which can be tuned to business needs.

**Model satisfies assumptions** We claim that this semi-supervised autoencoder model is able to effectively and robustly satisfy the anomaly detection problem constraints, and thus may be both more robust and more expressive than current models.

1. *Skewed class sizes*: while current models require balanced training class sizes (and thus hand-crafted sampling techniques), the semi-supervised approach does not, because it exclusively models the majority class.

2. *Concept drift*: current models require frequent re-training and hand-tuned temporal weighing to accurately model the volatile distribution of target-class samples. The semi-supervised approach does not, because it exclusively models the majority-class data (a fairly stationary data distribution)

3. *Model expressiveness*: current models are shallow, and thus have limited expressiveness. The autoencoder model admits arbitrary depth and thus arbitrary expressiveness (ability to model arbitrarily complex data distributions)

**Additional considerations** Another significant benefit of the semi-supervised approach is that the model does not require manual labeling of target-class transactions, a costly and often imperfect practice.

Note that the reconstruction error loss function offers flexibility, as it is fully tunable in two respects. First, the error metric can tuned to any problem (we can use l2, cross entropy, Hellinger distance, cosine similarity, etc). Second, since our loss is defined over feature values (rather than probabilities) we have the flexibility to attribute relative importance to features of interest, without influencing the model's representation of those features, by weighting each index in the feature vector by its relative importance. This measure offers greater control over the model to the business units, without sacrificing in expressive power.

## Recurrent Semi-supervised Autoencoder Model

We introduced a semi-supervised autoencoder capable of modeling stationary anomaly detection problems. While effective, the model was fundamentally unable to model temporally-dependent features. Here we expand on the static model, such that it can learn temporal anomaly-detection patterns. We extend the semi-supervised static autoencoder to learn temporal embeddings by parameterizing the encoder and decoder networks as LSTMs.

**LSTM Autoencoders** As explained in [1], autoencoders attempt to learn the mapping from an input to itself. A naively architected autoencoder would learn the identity function. By creating a bottleneck in the architecture, the model learns the optimal way to compress the information into a simplified representation. Autoencoders offer flexibility in that the encoder and decoder networks are black boxes, parameterizable by any neural network. Hence, we can parametrize the encoder and decoder with recurrent neural networks, if we aim to learn representations of sequences. As shown in [fig1], at time $t = i$, we feed the encoder LSTM with element $i$ of the sequence. At each timestep $t$, the LSTM encoder returns an output vector $z$ in latent space. The latent vector at time $t = T$ represents the entire sequence in latent space. Notice that this vector is no longer temporal (in fact, it has no notion of sequence length). We can think of this latent vector $z$ as the temporal input sequences embedding in an atemporal vector space $\mathcal{Z}$. To reconstruct the sequence from this embedding, we make $T$ copies of $z$ (for a sequence of length $T$). In order to reconstruct the original input sequence, we then pass a copy of $z$ as input to the LSTM decoder at each timestep.

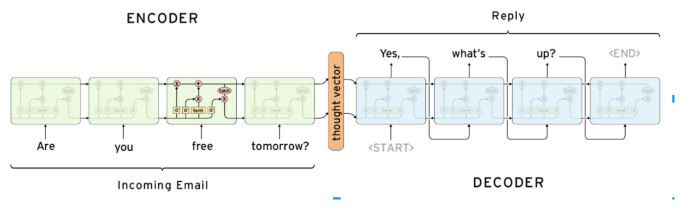

Figure 4: Recurrent autoencoder network

In neural language models, the latent vector $z$ (thought vector) can be interpreted as a sentence-level embedding. Generically, we can think of $z$ as a sequence-level embedding, for any arbitrary sequence encoding.

**Semi-supervised LSTM Autoencoder Model** In the same style as [1], we train this autoencoder in a semi-supervised manner in order to learn the distribution of majority-class user behavior.

We train the model on a set of majority-class sequences (with sequence-level labels). At test time, we evaluate a set of unseen sequences, and rank them by decreasing reconstruction error.

With respect to modeling sequences, we can interpret this semi-supervised training regimen as modeling the temporal distribution of majority-class transaction sequences. We expect the target-class sequences to differ from this temporal distribution in latent space, and consequently, in their reconstruction.

## Deep Semi-Supervised Embeddings Model

The proposed Deep Semi-Supervised Embeddings model architecture is visible in Figure 1 — to which the reader should refer throughout the following sections.

**Architecture Components** The proposed architecture (pictured in Figure 1) includes:

- An Encoder (Deep Neural Network)
- A Decoder (Deep Neural Network)
- A Classifier (Deep Neural Network)

**Training** First, we train an autoencoder via semi-supervised learning, on data from the majority class. We then use the trained autoencoder to measure reconstruction errors for each sample in the full training set (both majority and minority). We then discard any samples whose reconstruction error falls below a filter threshold constant $\gamma$. We map the remaining samples (whose reconstruction error exceeds $\gamma$) to latent space $\mathcal{Z}$ by passing them through the trained encoder. Finally, we train a supervised network with the remaining samples living in $\mathcal{Z}$.

To classify a new sample $x$ at test time, we encode the point into $\mathcal{Z}$, and then classify the resulting point with the trained supervised network.

These are the algorithms for training the proposed model, and for classifying a new point:

**Algorithm 1** Training Full Architecture

```
1: procedure TRAIN($X, Y$)                          ▷ train & return model
2:     $ae.train(X_{majorityClass})$
3:     $reconErrors \leftarrow ae.evaluate(X_{full})$
4:     $anomalies \leftarrow X.filter(\gamma, reconErrors)$
5:     $supModel.train(anomalies)$
6:     return $ae.encoder, supModel$
```

**Algorithm 2** Classifying New Sample

```
   procedure CLF($encoder, supModel, x$)          ▷ binary clf
2:     $z \leftarrow encoder.evaluate(x)$
       $pred \leftarrow supModel.evaluate(z)$
4:     return $pred$
```

For the more mathematically inclined reader, we can understand the encoder as a mapping (1) to a manifold $\mathcal{M}$ in latent space $\mathcal{Z}$ (the compressed representation in Figure 1).

$$Encoder_{ss} : X_{majorityclass} \rightarrow \mathcal{Z} \tag{1}$$

We then pass the entire training set through the autoencoder ("encoder-decoder network") whose reconstruction output is denoted $\hat{X}$ (2).

$$Decoder(Encoder) : X_{full} \rightarrow \hat{X}_{full}. \tag{2}$$

For each sample, we compute the reconstruction error (3). We discard the set of samples $x_i$ satisfying $R_i < \gamma$.

$$R_i = ||x_i - \hat{x_i}||_2^2. \tag{3}$$

We pass the filtered set of samples $\{x_i | R_i \geq \gamma\}$ through the encoder $Encode_{ss}(X_{\{i|R_i \geq \gamma\}})$, resulting in a distribution $Z_{embedded}$ of this set of samples in latent space $\mathcal{Z}$. Finally, we learn the posterior distribution $P_{supervised}(Y|Z_{embedded})$ with a supervised network.

In its entirety, the architecture learns the posterior distribution of class labels shown in (4).

$$P_s(Y|Encoder_{ss}(\{x_i|R_i \geq \gamma\}) \tag{4}$$

**Modularity** Next we describe some of the atoms that compose the full model, and explain why we would use them.

First, the encoder and decoder networks can be parametrized by any type of neural network, and thus offer flexibility in which types of data can be modeled. For example, if modeling a simple data distribution, we could use a canonical shallow fully-connected network for the encoder, decoder, and the supervised model. However, if we were modeling sequential data, we could use an LSTM encoder, an LSTM decoder, and a deeper fully connected architecture as the supervised model (we would not parametrize the supervised model with a recurrent architecture, because data distributions in $\mathcal{Z}$ are atemporal). This LSTM autoencoder closely resembles a Seq2Seq model (Sutskever, Vinyals, and Le 2014). Alternatively, If we were modeling images, or other data whose features have spatial dependency, we could parametrize the encoder, decoder, and supervised network with convolutional networks.

In short, we can tailor each component of this architecture to the type of data present in the dynamic targeted anomaly detection problem. Exploration of other network types that could be applied to problems involving other data types is left as an exercise to the reader.

**Implementation** For the sake of reproducibility, we describe how to implement the model in code.

The model can be easily implemented with the help of any computational graph engine, such as TensorFlow, Caffe, Theano, PyTorch, or even Keras. Networks can be trained using any popular objective functions (typically mean squared error for regression, and cross-entropy for classification), and optimized with any choice of gradient-descent based optimizer (SGD, Adam, RMSprop, etc.). When training the autoencoder, we recommend using large batch sizes.

Hyperparameters which need to be tuned include:

- At the full model level:
  - the dimensionality of $\mathcal{Z}$ (concretely, this is the dimension of the vector in the autoencoders compressed representation)
  - The filter threshold $\gamma$.
- At the component level:
  - Any hyperparameters to neural networks composing the encoder, decoder, and supervised network (network type, topology, and depth, layer types, and hyperparameters for particular layers, such as kernel sizes for convolutional layers)

## Example and Intuition

To understand this model intuitively, consider again John Smith's case. His behavior in Mint yesterday strayed from the expected majority class behavior. However, John's behavior is a perfectly valid use of Mint, and by no means fraudulent. If we were modeling John's behavior with a semi-supervised autoencoder, his behavior would likely be flagged as fraudulent, and Mint may have lost a customer. We claim that the model presented here has the capacity to classify John's behavior as benign.

## Why it works

We expect the model to be effective for two reasons:

1. We stretch the neighborhood of space in which the majority class anomalies and target class data live, thus giving ourselves greater discriminative capacity in this new transformed space $\mathcal{Z}$.
2. We remove the distributions of expected majority class behavior from question, thus allowing the supervised model to focus specifically on the task of distinguishing between the expected majority class members and the target class members, a far simpler task than distinguishing between the majority class (both anomalous and expected distribution) and the target class.

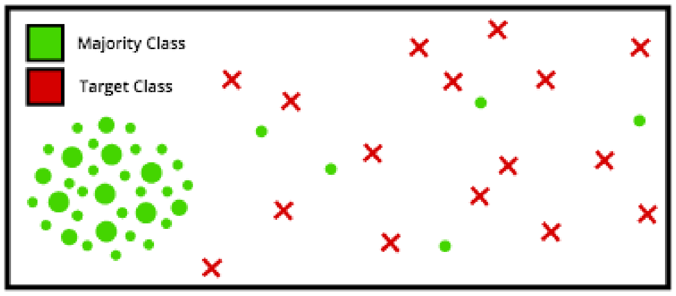

Figure 5: *Full training set in $\mathcal{X}$ space. Notice that in $\mathcal{X}$ space, the expected majority class behavioral data forms a clean distribution. However, there exist majority class anomalies interspersed among the target class samples. The goal of this model is to differentiate between the majority class anomalies (green points outside the cluster) and the target class samples (red points).*

At a high level, can think of the semi-supervised autoencoder models as learning the distribution of expected majority class behavior, and evaluating unseen points by their distance from this distribution. Given majority class anomalous points may significantly stray from the mean of this distribution, we do not expect the model to have the capacity to understand them as belonging to the majority class (see Figure 5).

We understand the aforementioned semi-supervised autoencoders to be learning the encoder:

$$Encode_{ss} : X_{majorityclass} \to \mathcal{Z}, \tag{5}$$

and the manifold $\mathcal{M}$ within $\mathcal{Z}$ which best represents data from the majority class. In the semi-supervised autoencoders by defining our target class probability as the extent to which behavior strays from the mean of the majority class, we sacrifice the capacity to delineate between majority class anomalies and the target class.

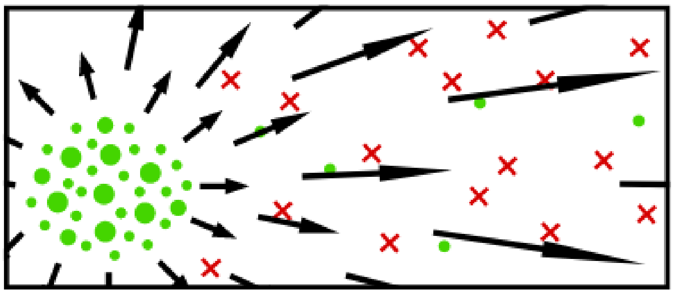
Figure 6: *Stretching of the space via embedding (the learned mapping from $\mathcal{X}$ to $\mathcal{Z}$). We represent the stretching of the original space around the expected majority class distribution (note that while the expected majority class is depicted above for reference, this distribution is NOT embedded, because it is first removed by the $\gamma$ filter). Alongside learning to project points from the expected majority class distribution onto the neighborhood of manifold $\mathcal{M}$, the Encode embedding learns to stretch the space outside this neighborhood in scale proportionate to its distance from $\mathcal{M}$ (notice the arrows grow in length as we get further from the green cluster center). We achieve a resulting increase in distance between points from outside the expected majority class distribution. In the diagram, vector direction represents the direction in which the space is "stretched", and vector length represent the degree to which space is "stretched".*

We claim that the embedding learned by the Autoencoder will separate out the expected majority class samples from rest of the samples, as shown in Figure 6. We must understand how this happens in order to understand how it motivates our new model architecture.

We understand the reconstruction error $R_i$ for any sample $x_i$ as a metric representing that points distance from the mean of the expected majority class distribution. When we discard any samples whose autoencoder reconstruction error falls below the $\gamma$ reconstruction error threshold, we are effectively removing the distribution of expected majority class samples from the space of all samples.

Instead of distinguishing between majority class samples (both anomalous, and from the expected distribution) and target class samples, we simply need to distinguishing between anomalous majority class user behavior, and target class user behavior.

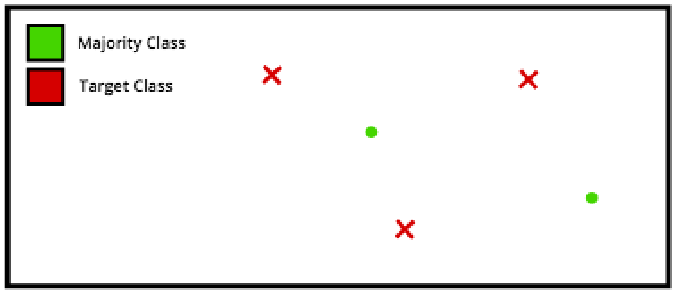

Figure 7: *The filtered, embedded points in $\mathcal{Z}$ space. To achieve this representation, we pass the $\gamma$ filter over all training samples, filtering out samples by reconstruction error (removing any points near the distribution of the expected majority class). We then project the remaining points, resulting in well-spread out points in the figure above. Note that the expected majority class distribution has been discarded.*

The semi-supervised autoencoder learns to optimally embed the training set. Since the training set consists exclusively of majority class samples, the vast majority of which belong to the expected majority class distribution, the embedding learned is an embedding on the expected majority class samples. We can imagine that the nonlinear principal components best representative of the expected majority class samples will be highlighted in latent space. Consequently, any samples that do not contain samples without those components will be drawn away from the neighborhood of normal samples in latent space 6. A good way to think of this is to understand that the manifold $\mathcal{M}$ learned by $Encode$ is concentrated sits in a neighborhood of $\mathcal{Z}$ space. The learning to map samples which resemble the expected majority class to this neighborhood of $\mathcal{Z}$, the embedding simultaneously learns to map samples which do NOT resemble the expected majority class to other neighborhoods of $\mathcal{Z}$ space distant from $\mathcal{M}$s neighborhood. In this way, the mapping learns to stretch the remainder of $\mathcal{Z}$ space.

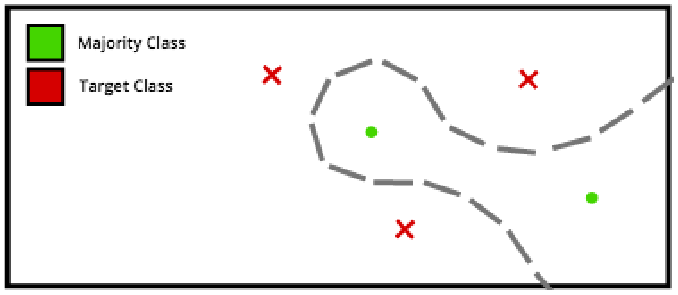

Figure 8: *We then train a supervised classification algorithm to differentiate exclusively between majority class anomalies and the target class, in latent space $\mathcal{Z}$. Now this task is significantly less complex, due to (1) Expansion of the space $\mathcal{Z}$, and (2) Removal of the expected majority class distribution.*

By stretching the space outside the expected majority class distribution, the embedding separates all samples outside the expected majority class distribution from the expected majority class distribution itself. Note that while this stretching effect is evident in the full encoder-decoder autoencoder reconstruction scheme, the stretching (relative to distance from the expected cluster) effect is largely achieved in the encoder alone, and thus we still achieve the stretching and separation effect via the embedding alone.

Finally, our resulting space is a “stretched” version of the original space, and does not contain the expected majority class distribution. We can now effectively learn (in a supervised manner) a decision boundary delineating between the expected majority class and the target class (See Figure 8).

## Algorithmic Extensions

We understand the importance of the following extensions to the models architecture:

1. Ability to model multiple levels of hierarchy. By stacking multiple semi-supervised autoencoders, wherein the $i$-th autoencoder trains, then embeds the full training set, and the $i+1$-th autoencoder uses this embedded set as its own training set. The final autoencoders reconstruction is then used as a $\gamma$ filter. A final supervised model is trained on the set of samples (embedded in the last autoencoders latent space) which pass through this $\gamma$ filter. Alternatively, we can filter iteratively at each level of depth, with a corresponding $\gamma_i$ filter at each hierarchy level $i$. With this architecture, we hope to gain the ability to model problems whose majority class forms many expected distributions, yet nonetheless contains outliers. We acknowledge that data from many applications of our architecture likely have a majority class divided amongst multiple expected classes, and thus highlight the potential for this promising extension of the proposed algorithm (Ronneberger, Fischer, and Brox 2015).
2. Ability to model alternatively-shapen expected majority class distributions. We recognize that our choice of reconstruction error (objective function for the autoencoder) presupposes some degree of knowledge regarding the shape of the distribution of expected majority class features we are seeking to model (for example, mean-squared error presupposes well defined first and second moments, and is most interpretable under Gaussian distribution assumptions). We thus motivate alternative distance metrics as reconstruction error metrics, which may capture varying measures of distance from the expected majority class distributions. Metrics include:
   - Hellinger distance
   - Wasserstein distance (earth movers distance)
   - Weighted MSE (attributes feature-level importances)
3. Ability to generate synthetic data from the expected majority class distribution, make stronger assumptions about the supervised model, and gain interpretability by being able to analyze the learned manifold in $\mathcal{Z}$. We motivate the use of a Variational Autoencoder as the semi-supervised autoencoder architecture. This allows us to push the latent space representation towards a Gaussian distribution, and thus (1) Have stronger assumptions both about the data passed to the model, (2) Be able to generate realistic synthetic data by passing a randomly sampled point from the surface of this manifold to the decoder, and (3) Gain interpretability of the manifold (by analyzing its tangent space) and thus gain a richer understanding of the distribution learned by the autoencoder. Finally, we motivate use of a new class of autoencoder that draws the latent distribution towards that learned by running Topological Data Analysis on the majority class. In this manner we hope to learn a more geometrically robust latent space representation, and by doing so, more intelligently demark the boundaries of the expected majority class distribution, and more intelligently “stretch” the space.

## Practical Applications

We see application of this model is various practical applications fitting the problem domain:

- Fraud Detection
  - In static datasets
  - Temporal datasets
- Outlier detection in Tax Form images
  - CNN network parametrization
- Clickstream behavioral anomaly detection
  - Detecting abandonment from rare behavior
  - Detecting subscription from rare behavior
- Transaction Categorization
  - For identifying rare transactions

## Conclusion

We introduced a novel deep learning algorithm able to effectively solve any “dynamic targeted anomaly detection” classification problem, as defined by its skewed class sizes, concept drift in the target class, and anomalies in the majority class.

## References

Bengio, Y.; Lamblin, P.; Popovici, D.; and Larochelle, H. 2007. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, 153–160.

Chen, J.; Sathe, S.; Aggarwal, C.; and Turaga, D. 2017. Outlier detection with autoencoder ensembles. In *SIAM Conference on Data Mining*.

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

Dai, A. M., and Le, Q. V. 2015. Semi-supervised sequence learning. In *Advances in Neural Information Processing Systems*, 3079–3087.

Ganin, Y., and Lempitsky, V. 2015. Unsupervised domain adaptation by backpropagation. In *International Conference on Machine Learning*, 1180–1189.

Li, J.; Luong, M.-T.; and Jurafsky, D. 2015. A hierarchical neural autoencoder for paragraphs and documents. *arXiv preprint arXiv:1506.01057*.

Lyudchik, O. 2016. Outlier detection using autoencoders. *CERN. Geneva. EP Department, Computing and Computers*.

Pan, S. J., and Yang, Q. 2010. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering* 22(10):1345–1359.

Ronneberger, O.; Fischer, P.; and Brox, T. 2015. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, 234–241. Springer.

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, 3104–3112.

Weston, J.; Ratle, F.; Mobahi, H.; and Collobert, R. 2012. Deep learning via semi-supervised embedding. In *Neural Networks: Tricks of the Trade*. Springer. 639–655.

Wu, Y.; Schuster, M.; Chen, Z.; Le, Q. V.; Norouzi, M.; Macherey, W.; Krikun, M.; Cao, Y.; Gao, Q.; Macherey, K.; et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.