August 8, 2024

A Semi-Supervised Approach to Anomaly Detection for Tax Compliance

Leo Corman[1], Jason Bono[1], Catherine Acton[1], Danielle Gewurz[1], **Evan Schulz**[2]

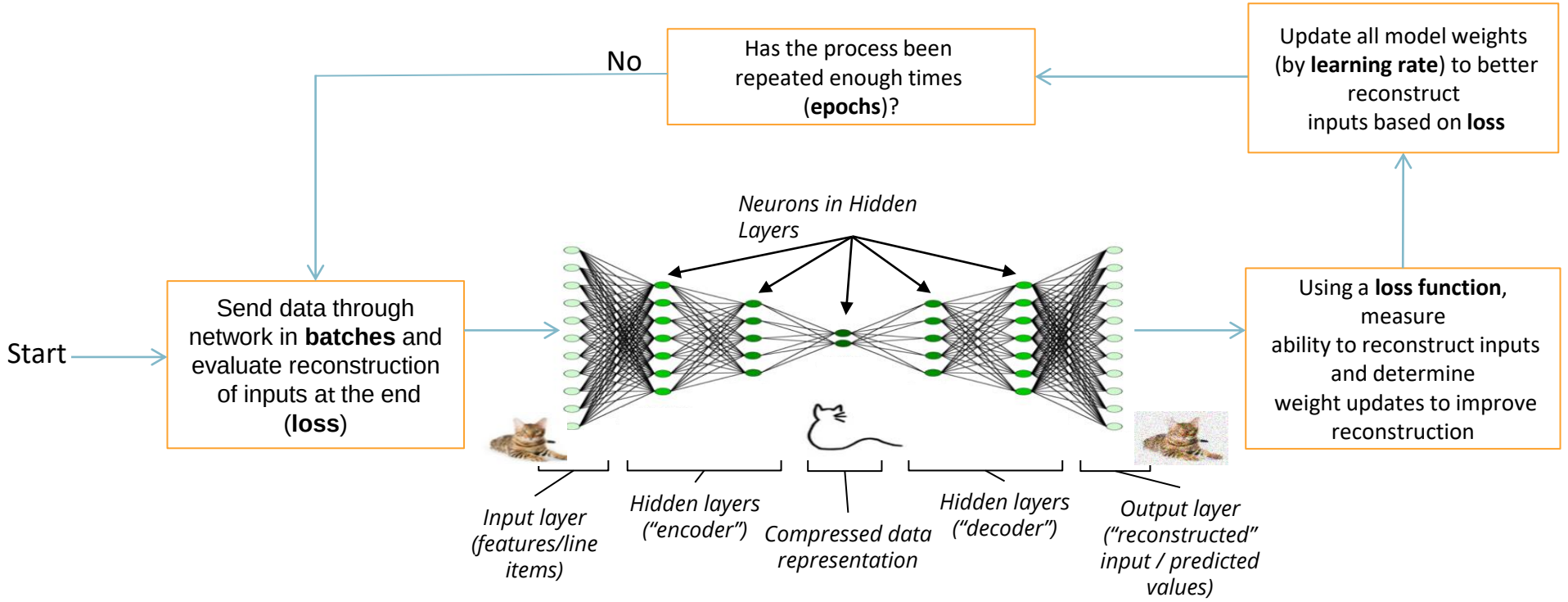
[1]*Deloitte Consulting LLP*

[2]*Internal Revenue Service – Research, Applied Analytics, and Statistics*

Background

Autoencoders can capture complex patterns in data

- Goal is to improve reconstruction with each iterative update to minimize the loss function



Background

Autoencoders can be used for anomaly detection in tax data

- For tax form data:
  - Input = Line item values reported by the tax filer
  - Output = Predicted (“reconstructed”) values for those line items
- The compressed representation of the data can be thought of as capturing “typical” patterns and relationships in the tax data
- A large difference between reported and predicted value (reconstruction error) signals an anomaly and potential non-compliance at the line item level
- Line item reconstruction errors can be aggregated to produce a return-level score
- See previous research (Acton et al., “Anomaly Detection on Sparse Data with Autoencoders,” 2023)

Key Additions to Autoencoder Approach

Expanding on previous research, we introduced new elements to the autoencoder approach, including:

- **Semi-supervised loss function**: Incorporates labeled data into the training process
- **Two-model scoring**: Produces a combined score from two semi-supervised models that use labels differently

Semi-Supervised Loss Function

New loss function allows for the inclusion of labeled data with unlabeled data during training

- $Batch\ Loss = \sum_{i=1}^{n} MSE(x_i) + \eta \sum_{j=1}^{m} MSE(\tilde{x}_j)^{\tilde{y}_j}$
- Based on loss function from Ruff et al. (2020)
- MSE is sparse Mean Squared Error: $MSE = \frac{1}{k_{filled}} \sum_{l=1}^{k_{filled}} (populated\ line_l - predicted\ line_l)^2$
- x_i are unlabeled returns, $\tilde{x}_j$ are labeled returns
- $\tilde{y}_j$ are labels for $\tilde{x}_j$ (1 for compliant returns, -1 for non-compliant returns)
- $\eta \geq 0$, allows for weighting labeled data relative to unlabeled data

Two-Model Scoring Approach

Two-model scoring approach combines two semi-supervised autoencoder models, a "flipped" model along with the original

- The "flipped" model is the same as the original (same training data, parameters, loss function, etc.), but with labels reversed for training
- This incentivizes the "flipped" model to reconstruct non-compliant returns well and compliant returns poorly; The result is one model (M^+) that flags non-compliance as anomalous and another (M^-) that flags *compliance* as anomalous.
- Flipping the labels creates a non-trivial difference between M^+ and M^- batch loss functions:
  - Positive and negative labels affect the loss function asymmetrically and the unlabeled data still influences model training
- $Two\ Model\ Score_j = \frac{M^+(\tilde{x}_j) - M^-(\tilde{x}_j)}{M^+(\tilde{x}_j) + M^-(\tilde{x}_j)}$, for return $\tilde{x}_j$
  - The highest-scoring returns are those with relatively large differences between M^+ and M^-

Data

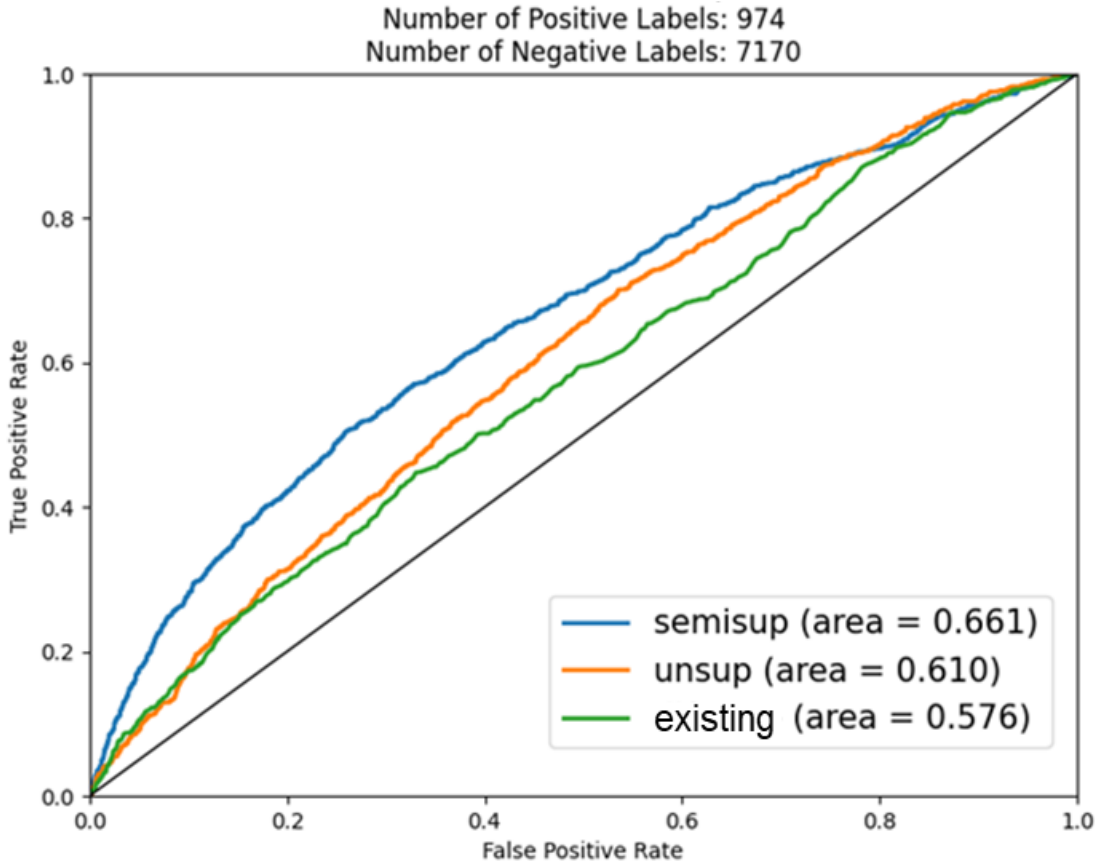
Semi-supervised model training requires both labeled and unlabeled data

- Unlabeled data generally comprises all Form 1040 returns filed for a single tax year
- Two sources of labeled data: National Research Program (NRP) and “operational” examinations
- Features come from 1040 main form and associated schedules (~260 line items)
- Form 1040 population is separated into strata for training; performance was evaluated within and across strata
- Within each training stratum, row and column filtering is used to help the model capture salient patterns in the data:
  - Remove columns (line items) that are filled infrequently; remove rows (returns) with insufficient populated line items
- Data pre-processing includes a cube root transformation (optional) and Maximum Absolute Value scaling to make each line item range from 0 to 1

Results

Semi-supervised learning improves performance over a fully unsupervised autoencoder as well as the existing operational benchmark model

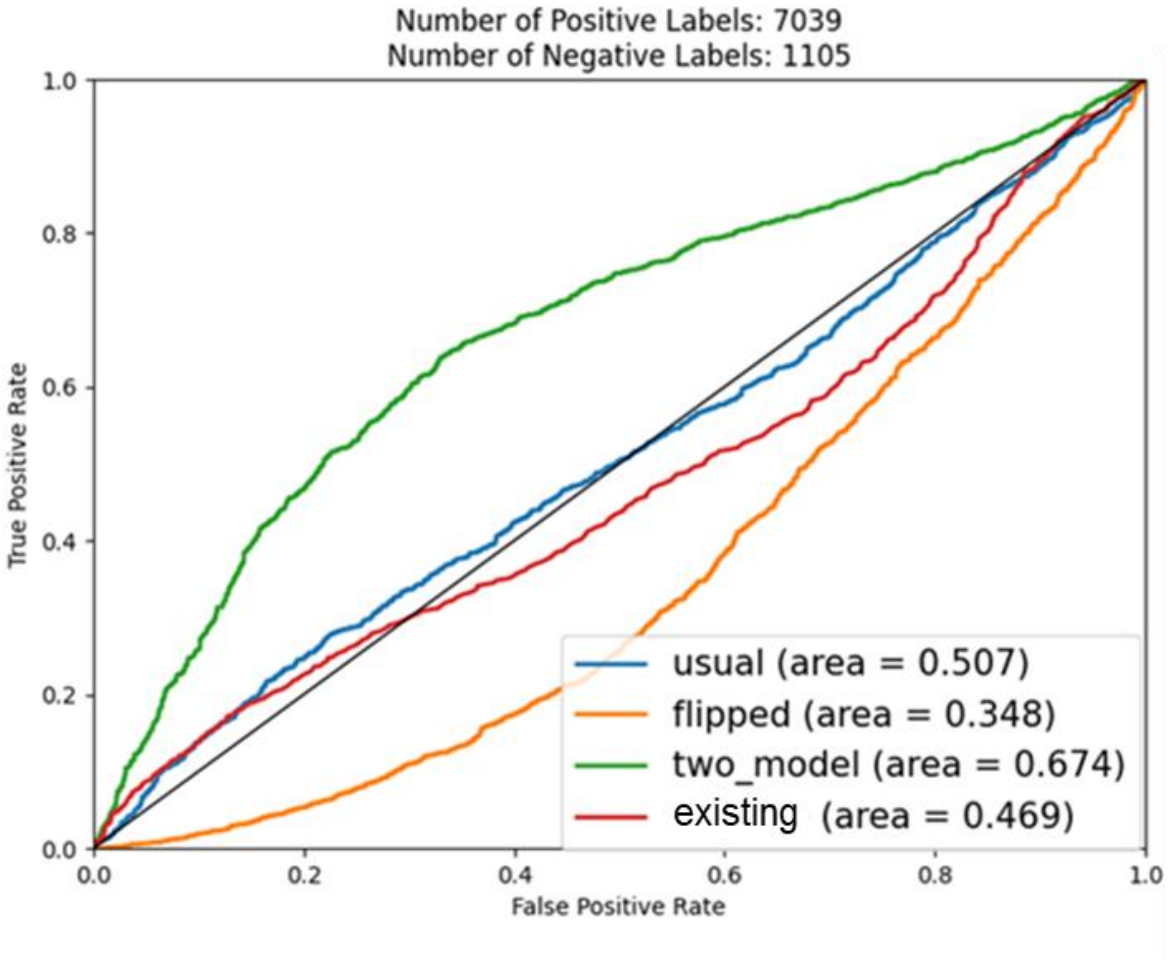
- The figure displays performance in one stratum
- The semi-supervised autoencoder and unsupervised autoencoder have the same parameter values except for $\eta = 0$
- None of these models involve two-model scoring



Results

Two-model scoring provides a boost over non-ensemble models

- The figure displays performance in one stratum
- Note that it is possible for both the “usual” (M^+) and “flipped” (M^-) models to have AUC less than 0.5 and still achieve a two-model AUC well above 0.5.



Results

Varying η produces only moderate changes in performance; stronger performance is associated with smaller η values

- Smaller values for η reduce the likelihood of overfitting the model to the labeled data
- In the table, the highest AUC value achieved within each selected stratum is highlighted (in the case of ties, the lowest value for η is highlighted)

AUC by Stratum for Different η Values

Stratum	$\eta = 0.001$	$\eta = 0.01$	$\eta = 0.1$	$\eta = 1$	$\eta = 10$
E	0.59	0.64	**0.65**	0.65	0.65
I	0.53	0.60	0.61	**0.65**	0.61
J	0.61	**0.63**	0.63	0.62	0.61
K	0.55	0.57	**0.58**	0.58	0.55

Results

Autoencoders consistently outperform the existing operational benchmark model across strata

- The two-model approach achieves the highest AUC value most frequently across strata

AUC by Stratum for Autoencoders vs. Benchmark Model

Stratum	M^+ Model	Two-Model Ensemble	Benchmark
A	**0.60**	0.54	0.49
B	0.51	**0.59**	0.53
C	**0.59**	0.58	0.58
D	0.47	**0.56**	0.52
E	0.51	**0.68**	0.47
F	0.52	**0.56**	0.54
G	**0.62**	0.59	0.56
H	0.54	**0.57**	0.51
I	0.54	**0.62**	0.50
J	0.52	**0.62**	0.51
K	0.50	**0.59**	0.53

Summary of Findings

Semi-supervised methods show encouraging results

- **Semi-supervision improves performance**: The semi-supervised autoencoder consistently outperforms the unsupervised autoencoder with the same parameters
- **Two-model ensemble improves performance**: The two-model score often provides a significant boost in AUC over M^+, although the boost is more consistent for certain strata than others
- **Performance is favorable relative to benchmark**: The semi-supervised autoencoder models (both M^+ and two-model) consistently outperform the existing operational model
- **Performance generalizes**: The semi-supervised autoencoder models perform well across strata and across different datasets
- **Cube root transformation improves performance**: Transforming each line item using the cube root, combined with corresponding adjustments to the learning rate and weight decay parameters, provides a solid boost in AUC

References

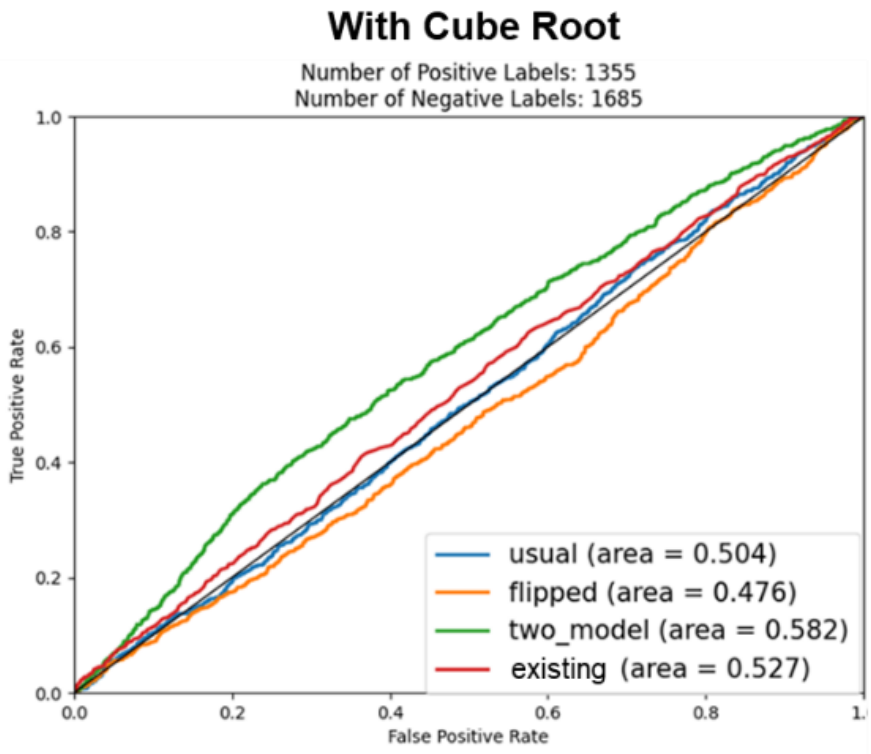
- A. S. Parker, D. Gewurz, and W. J. J. Roberts. “Quality and Validity Testing of Sparse Form Data using Gaussian Mixture Models," JSM Proceedings, Social Statistics Section, 2018.
- A. S. Parker, D. Gewurz, and W. J. J. Roberts. “Recommender Algorithms for Form Anomaly Detection," JSM Proceedings, Government Statistics Section, 2020.
- C. Acton, L. Corman, J. Bono, D. Gewurz, C. Walsh, and E. Schulz. “Anomaly Detection on Sparse Data with Autoencoders,” JSM Proceedings, 2023, DOI: 10.5281/zenodo.10001050.
- N. Merrill and A. Eskandarian. “Modified Autoencoder Training and Scoring for Robust Unsupervised Anomaly Detection in Deep Learning,” IEEE Access, vol. 8, pp. 101824-101833, DOI: 10.1109/ACCESS.2020.2997327.
- L. Ruff, R. Vandermeulen, N. Görnitz, A. Binder, E. Müller, K-R. Müller, and M. Kloft. “Deep Semi-Supervised Anomaly Detection,” International Conference on Learning Representations, 2020, URL: https://doi.org/10.48550/arXiv.1906.02694.

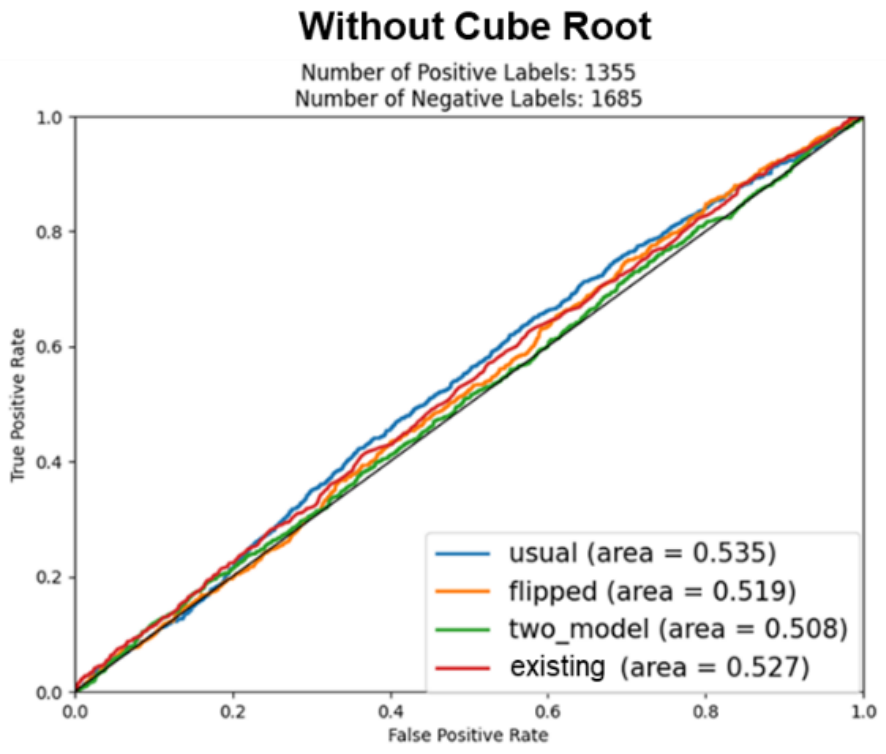
Appendix

Results

Applying a cube root transformation provides a noticeable boost in performance

- The figures display performance in one stratum





Semi-Supervised Loss Function

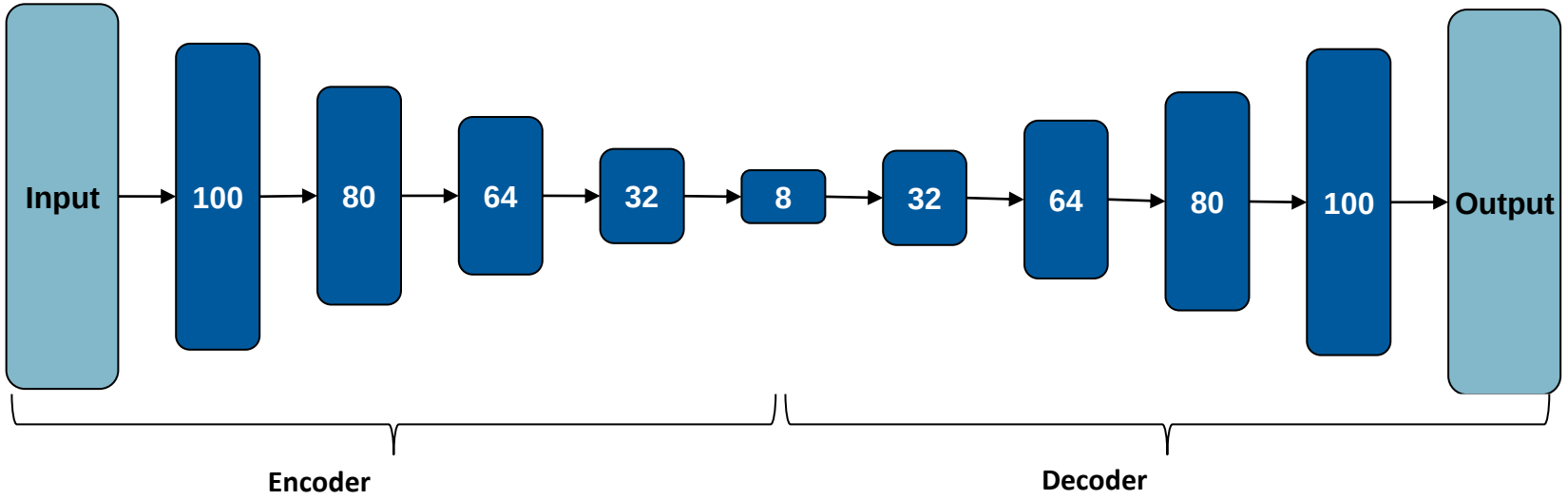
Further modifications to the loss function were tested

- Modified version: $Batch\ Loss = \sum_{i=1}^{n} MSE(x_i) + \sum_{j=1}^{m} (\eta + \dot{S}(\tilde{x}_j)) MSE(\tilde{x}_j)^{\tilde{y}_j}$
- Consider a continuous "success" metric, $S(\tilde{x}_j)$, for return $\tilde{x}_j$, where $S(\tilde{x}_j) = 0$ for compliant returns
- $\tilde{y}_j = \{-1 \text{ if } S(\tilde{x}_j) \geq d;\ 1 \text{ otherwise}\}$, where d is some threshold for case success
- $\dot{S}(\tilde{x}_j)$ is a version of $S(\tilde{x}_j)$ scaled to have magnitude roughly proportional to the chosen value for η

Autoencoder Network Structure

Autoencoder structure is relatively simple, with linear layers

- Five hidden linear layers for both the encoder and decoder, with [100, 80, 64, 32, 8] neurons per layer
  - This means the middle layer has 32*8 + 8 parameters
- SELU for hidden layer activation functions
- Tanh for output activation function



Autoencoder Parameters

Performance was evaluated for many parameter combinations

- The following parameters were controlled the use of labeled data during model training
  - Varying *eta* was given particular focus

Parameter	Definition	Exploration
train_with_thresh	Whether to only train on labeled returns with Dollars Recommended >= *threshold*	Setting to True was found to increase performance for both “change” and “above threshold” cases for the original model, but results were mixed with the two-model approach
eta	Weight on labeled data in loss function	Tested values ranging from .001 to 1000; strong performance was still achieved with small *eta*, while larger *eta* values did not change performance significantly; most frequently *eta* = 1
scale_eta	Whether to scale *eta* based on Dollars Recommended (i.e., returns with higher Dollars Recommended get higher weight); implemented as $\text{eta} + \log_{10}(dollars + 1)$	Setting to True did not significantly change performance when *eta* = 1, although when *eta* = 0, performance was similar to setting *eta* = 1 and *scale_eta* = False

Autoencoder Parameters

Performance was evaluated for many parameter combinations

- Additional modeling parameters were tested, as noted in the table below
  - The number of epochs and the neural network structure were given particular focus (* indicates a parameter was tested in more detail in Acton et al. (2023))

Parameter	Definition	Exploration
epochs	Number of times all data will be passed through the network and parameters will be updated through backpropagation	Tested values ranging from 1 to 100; strong performance was achieved with as few as 5-10 epochs
batch_size*	Number of returns to include in each training batch (to make computations more manageable)	Fixed at 10,000
learning_rate*	Magnitude by which to adjust model parameters during each training iteration	Fixed at 8e-5 for earlier experiments, then re-baselined with cube root and set to 1e-3
weight_decay*	Regularization method to penalize larger weights in the network	Fixed at 1e-8 for earlier experiments, then re-baselined with cube root and set to 1e-5
plt_threshold*	Used to prevent learning patterns from unlabeled returns that have reconstruction error above the threshold	Fixed at 1 for most experiments; values ranging from 0.7 to 0.9 did not significantly change performance
neurons_list	Controls number of hidden layers, number of neurons in each layer, and presence of dropout layers	Tested smaller bottleneck layer (i.e., 4 vs 8 neurons), addition of 20% dropout layer(s), and smaller number of hidden layers

Autoencoder Parameters

Performance was evaluated for many parameter combinations

- Other parameters were implemented to control which data is used for training/testing, how the data is processed, and how experiments are run
  - These parameters are considered fixed within a single “run” of the code; the other parameters can be varied within a single run

Parameter	Definition	Exploration
data_sources	Which data to use for unlabeled, train, and test sets	(see paper for details)
training_subsets	Which strata to train on and how to group strata for training	Tested all strata, with focus on larger or higher priority strata; also tested combining strata during training
number_of_trials	How many separate times to train a model for each parameter combination	Generally ranged from 1 to 5
min_col_density	Minimum number of times a line item must be populated to be kept for training	Varying from 100 to 1000 did not significantly change performance
cbrt_transform	Whether to apply cube root transformation to data	Setting to True improved performance noticeably, especially after re-tuning other parameters in conjunction