**AXA Telematics Challenge: Use of Autoencoders**

**Arjun Chakraborty**

**Georgia Institute of Technology**

**Introduction:**

This work investigates the performance of autoencoders in the domain of unsupervised anomaly detection. It explores the implementation of the algorithm to a publicly available dataset. Anomaly detection is one of the most popular topics in the field of machine learning. The goal of the algorithms designed for anomaly detection are concerned with finding data in a dataset that does not conform to a certain pattern. The idea behind this is to identify data points that are special in regards to their behavior, compared to data points that are considered to be ‘normal’.

Anomaly detection can happen in a supervised, semi-supervised and as well as in an unsupervised manner. Something like credit card fraud detection would be a semi supervised learning task. In this case, newly generated data points can be tested on whether they conform to a pattern or not. However, in the case of unsupervised anomaly detection, we do not know what a ‘normal’ data point looks like. The absence of labeled data leads to a different approach when it comes to finding outliers. In unsupervised anomaly detection, it is essential to assume that the number of ‘normal’ points far exceeds the number of anomalous points. This is very import as it helps the model to learn what is constituted as ‘normal’, therefore helping the model to detect deviations.

**Dataset:**

The AXA Driver Telematics dataset is a dataset that was released to the public in the form of a Kaggle challenge. It contains the logs of 2736 drivers. There is one designated folder for each driver, which contains 200 different trips in the form of CSV files. A single trip consists of a CSV file, consisting of two columns (representing x and y coordinates) and varying number of rows. Each row represents the driver’s position one second after the previous row. Each trip is anonymized, such that each trip starts at (x,y) = (0,0) and all the coordinates have been randomly flipped.

The key to finding the outliers is based on the fact that a varying and unknown number of trips have been added to folders generated by other drivers. The dataset is considered to be reasonably large with a size of 1.44 GB in compressed state. The reason for identifying anomalies in driving patterns

**Preparation of Data:**

The features need to be extracted that describe the driving behavior of the driver and also the road conditions. During this work, I have used 12 features. These have been stated below. For every driver, a feature matrix of 200 X 12 would be fed into the input layer of the autoencoder. The training/test split is 70/30.

* Length of each trip
* Average velocity over each trip
* Percentile velocity (Percentile: 5,10,50,75,85,95)
* Count of number of stops before a certain distance threshold
* Count of number of stops before a certain velocity threshold
* Ratio of total number of stops over the length of a particular trip
* Acceleration
* Percentile of Acceleration
* Heading angle
* Percentile of Heading angle
* Turning aggression (Speed\*Heading angle)
* Percentile of turning aggression

**Trip Segment Features**:

Another idea with regard to trip segment features is to identify common road segments between trips. This approach would help identify redundant trips. The Ramer-Douglas-Peucker algorithm would help in simplifying trajectories. However, due to time constraints, I have not been able to implement the algorithm right now.

**Implementation:**

**Autoencoder**:

It is a class of neural networks that are designed to perform dimensionality reduction and manifold learning. It helps in better representing unlabeled data. The training is done in a supervised manner. The simplest autoencoder consists of an input layer, hidden layer and output layer. It works on the intuition that the performance of the machine-learning algorithm is dependent on the features it is being applied to.

**Stacked Autoencoder:**

It consists of multiple layers of autoencoders in which the outputs of each layer are wired to the successive layer. It consists of a wider class of functions, which help, in dimensionality reduction. Each layer is trained at a time.

**Deep Learning Framework**: Theano

**Loss function:** Mean Squared Error (MSE)

**Activation Function used**:

1. **Sigmoid activation function**: I would generally be using the sigmoid activation function. As it is easy to implement, and Theano has a built in function for this particular activation function.
2. **ReLu Activation function**: This activation function is used when the number of hidden layers increases. It helps eradicate the vanishing gradient problem.
3. **TanH activation function**

**Algorithm for training the autoencoder**:

1. Initialize no. of epochs, learning rate
2. Initialize loss function
3. epoch = 0
4. While (epoch < no. of epochs)
5. epoch = epoch + 1
6. for minibatch = 1 to N
7. compute MSE
8. update parameters with respect to MSE using stochastic gradient
9. Update learning rate and momentum

(10) Store weights and biases for test set

**Experimental Design:**

1,2 and 3

No of Hidden Layers

128 and 256

Size of Mini Batch

0.1, 0.01 and 0.001

Learning Rate

100,500 and 1000

Hidden Layer nodes

1,2 and 5 percent

De-noising rate

Sparse and Random

Weight Initialization

**Training Sequence:**

**(1) Training:** Mean Squared error loss function

**(2) Optimization:** Stochastic Gradient Descent

**(3) No of epochs:** Chosen by the user

**(4) Learning rate:** Decayed by 0.985

**Evaluation of Results:**

*Testing out Autoencoder model:*

* The visualization of weights (unlike in the case of MNIST) is not possible in this case. Also, it is not possible to submit the results to Kaggle, to determine AUC. This is due to the time constraint
* The L2 norm of the reconstruction error for each trip of the training set is computed. The same procedure is carried out for the test set.
* The assessment of the L2 norms of the training and test set is done using Kolmogorov-Smirnov (KS) Test.
* The KS test gives a good indicator about the relationship between the distribution of the training and the test set. If the null hypothesis is not rejected, then the distribution may be the same. This indicates the algorithm may have generalized well. Otherwise, over-fitting may have occurred.

*Anomaly Detection:*

To detect anomalies:

* Train model properly. Mean reconstruction error should be reducing with no of epochs.
* Plot the reconstruction error using matplotlib.
* In the case of each driver, there are 200 trips. Filter out approximately 15 trips with highest reconstruction error. This helps in detecting a threshold.
* Use the same threshold to detect anomalies in the test. Using the KS test, we have already established that our autoencoder generalizes well.

**Challenges:**

1. It was difficult to gain insight into which features where necessary. The data itself gave out no information.
2. As visualization of weights was not possible, I had to come up with an algorithm to evaluate the working of the autoencoder.
3. I have had no experience with anomaly detection. While working on this challenge, I gained valuable insight.
4. Methodology of anomaly detection.

**Future Work:**

1. Test autoencoder model using the different parameters mentioned in the experiment design.
2. Implementation of denoising autoencoder
3. Use trip-matching techniques such as RDP algorithm.
4. Extraction of a larger number of features.