Anomaly Detection with Normality Modelling

Karan Jit Singh
Georgia Institute of Technology
College of Computing
karanjs@gatech.edu

***Abstract*—With almost every problem that involves similarity detection or categorical classification, the subject of classification in the collected data must be statistically significant for a model to perform relatively well. Such classification or similarity detection becomes much harder when the statistical presence of information of interest in the dataset is much lesser. We propose a different paradigm for such anomaly detection than typical classification and clustering techniques. Instead of explicitly classifying the anomalies, we focus on modeling the predominant normality of the dataset and hence endeavor to detect deviations in the form of anomalies. We further present an extension that suppresses false positives and false negatives from the normality modeling.**

***Keywords*—Machine Learning, Anomaly Detection, Unsupervised Learning, Representation Learning, AutoEncoders**

I. Introduction

In typical classifications and classification models, the specific classes of interest to be labeled usually predominate the data. In contrast, Anomaly detection techniques focus on labeling and classifying datasets where the occurrence of the subject is statistically rare. In specific scenarios where such occurrences in datasets are extremely rare, even traditional anomaly detection techniques fail to detect occurrences accurately. Additionally, for such scenarios, data labeling can prove even more difficult.

This project focuses on anomaly detection in time series problems. We make two assumptions about the dataset. First, we assume that most of the given data is normal and only a tiny percentage is abnormal. Second, we anticipate that abnormal data is statistically different from normal. According to these two assumptions, data groups of similar instances that appear frequently are assumed to be normal. In contrast, infrequent instances which are considerably various from the majority of the data are regarded anomalous. For our experimentation, we employ a different approach to anomaly detection. Instead of following in the path of traditional techniques and modeling the infrequent anomalies of interest in the dataset, we focus on models that train on the predominant "normal" scenarios of the dataset to detect deviations in the datasets to detect the anomalies.

One such problem in time series anomaly detection is Daphnet Freezing of Gait (FOG) [1]. The FOG dataset is devised to benchmark automatic methods to recognize gait freeze from wearable acceleration sensors placed on legs and hips. The dataset was recorded in the lab with an emphasis on generating many freeze events. Users performed three kinds of tasks: straight-line walking, walking with numerous turns, and finally, a more realistic activity of daily living (ADL) task, where users went into different rooms while fetching coffee, opening doors, etc.

For this project, we mainly work with proven time series ensembles such as neural networks, including LSTM and convolutional techniques and autoencoders. We experiment with both unsupervised methodologies for feature learning and semi-supervised learning for improving the results. The goal of the project is to train these models on time series datasets in order to detect anomalies and deviations in the time series.

II. Related Work

Anomaly detection is a vast field in machine learning. We surveyed the major techniques for general anomaly detection as well as anomaly detection in time series in the machine learning paradigm. Both categories contain machine learning techniques. The paper "Machine Learning Techniques for Anomaly Detection" [2] overviews the most general techniques and serves as our understanding for anomaly detection. The paper, however, only covers possible models from both supervised and unsupervised categories on anomaly detection without generalized experimentation. The most notable models that stand out for our problem set are unsupervised neural networks and One-Class SVMs. As described in the paper, these techniques do not need training data. Both these models are assumed as single class outputs that can be trained on a given dataset whose output can be interpreted as confidence intervals for the normality of the test input.

Another paper, "Anomaly detection in time series" [3], discusses techniques that are used in this project. Existing work with neural networks on Daphnet Freezing of Gait, DeepConvLSTM [4] detects anomalies in the said dataset using supervised neural networks to classify the anomalies. This paper is best fit to serve as our baseline for experiments with the FOG dataset.

III. Daphnet Freezing of Gait

[1] Presents a dataset Daphnet Freezing of Gait. Freezing of Gait (FOG) typically manifests as a sudden and transient inability to move. About 50% of all Parkinson's Disease patients regularly show FOG symptoms [5]–[7]. 10% of PD patients with mild symptoms and 80% of those severely affected regularly experience freezing. FOG occurs more frequently in men than in women and less frequently in patients whose main symptom is tremor [8]. PD Patients who experience FOG frequently report that their feet are inexplicably glued to the ground during the FOG episodes [9]. FOG is difficult to measure as it is highly sensitive to environmental triggers, cognitive input, and medication.

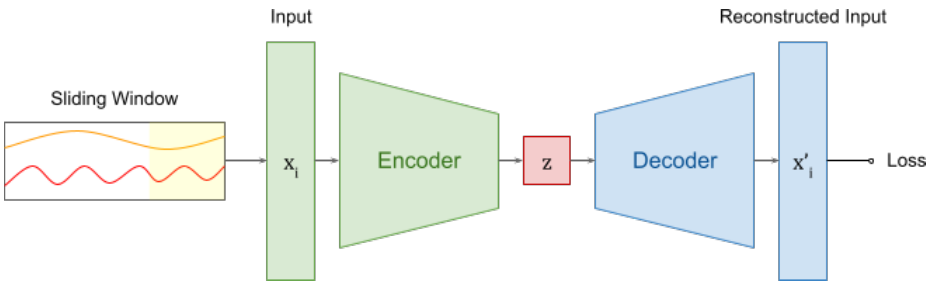

Fig 1. An AutoEncoder with LSTM layers for the encoder and decoder. The architecture takes a sliding wndow of input and outputs a reconstruction. This reconstruction error, or loss, is the L1 norm of the input and the reconstructed input.

Two sensors used to measure 3D acceleration were attached to the patients' leg; one at the shank, just above the ankle, and the other to the thigh, just above the knee. A third 3D-accelerations sensor was attached to the belt at the lower back of the patient. The acceleration sensors are 25×44×17 mm 3 in size and weigh less than 22 grams, including a rechargeable 300 mAh Li-ion battery with six h battery life. The acquired data is transmitted to the wearable computer over a wire-less Bluetooth link (64 Hz) for online data processing. The earphones are placed loosely around the patient's neck. The computing system produces a 1 Hz ticking sound starting whenever a FOG episode is identified and ending when the patient resumes walking. Fig 2. shows the wearable system worn by a patient.

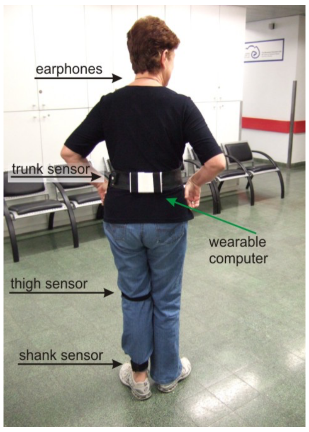

Fig 2. FOG detection and feedback assistant worn by one patient. Sensors are attached to the shank (just above the ankle) and the thigh (just above the knee) using an elasticized strap and Velcro. A third sensor is attached to the lower back to the same belt that the wearable computer is attached to.

IV. Normality Modelling with AutoEncoder

We implement an LSTM autoencoder for normality modeling. The autoencoder takes a sliding window input of the multivariate time series that pass through sequentially decreasing LSTM lasers until the smallest LSTM layer, i.e., the bottleneck; this sequence of layers forms the encoder of the architecture. The encoder compresses and learns a dimensionally reduced context of the input passed. This compressed context from the bottleneck is then passed through sequentially increasing LSTM layers that form the decoder. The decoder hence reconstructs the input. We calculate and reduce the L1 reconstruction loss to train the network. Training the autoencoder in such a way does two things:

1. In a dataset that is statistically dominated by a class of labels that we identify as normal. By training the autoencoder against its reconstructed input.

2. For statistically rare label classes, since the autoencoder doesn't train on such samples as much as it does on the *normality* in the data, the autoencoder's loss hence is much higher for such inputs. Fig 2. Shows the architecture described above.

We experiment with two significant philosophies of LSTM AutoEncoders:

1. In the first experiment, we use the hidden states of the LSTM from the bottleneck LSTM layer, which are then repeated n times, where n is the sliding window size.

2. In the second experiment, we use the time-series output of the bottleneck LSTM layer directly to the decoder.

To train the network, we used a sliding window of input size 24. We form a batch of 50 consecutive sliding windows. The network is then trained on the shuffled set of batches. We implement two LSTM layers of size 128 and 64 for both encoder and decoder. The bottleneck LSTM layer is chosen to be of size 8. Fig 3. Shows the loss of the network over 30 epochs and Fig. 4. shows the output of the autoencoder on one of the test samples.

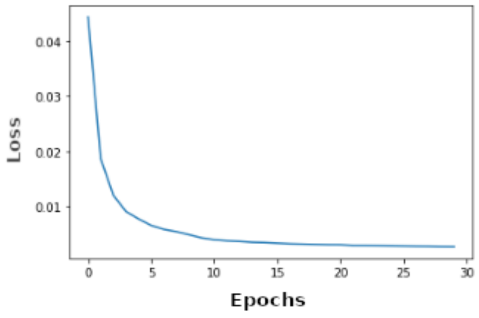

Fig 3. Mean loss over 30 epochs of training.

From Fig 4. we see that the reconstruction loss is much higher around anomaly regions. We also see that there are loss spikes in non-anomaly regions as well. This confirms our hypothesis that the model loss spikes on features in data it hasn't seen before.

For the two experiments with LSTM encoder context capturing philosophies, we found that using the hidden context from the LSTM proved to get better results. Fig 5.

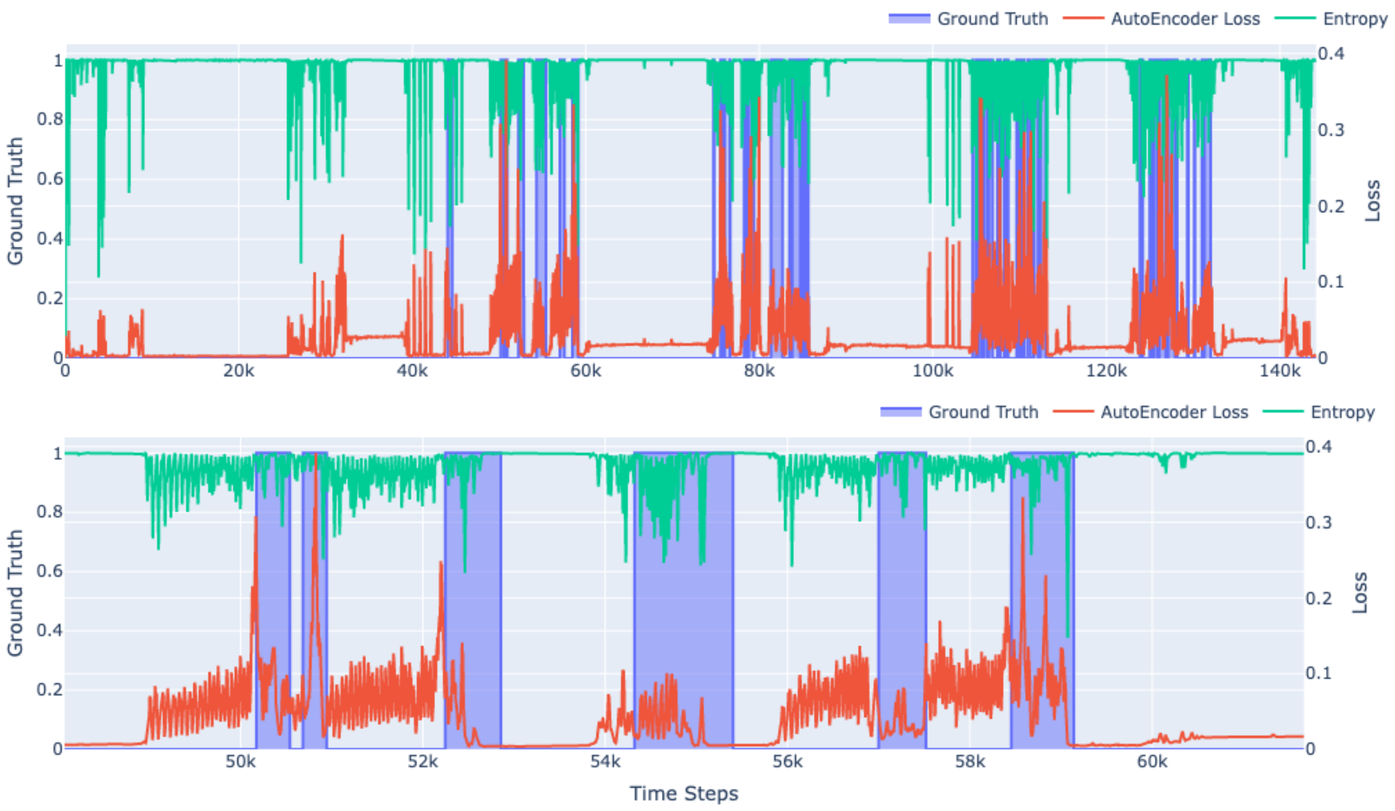

Fig 4. Output of the AutoEncoder for a test sample. The orange line represents the reconstruction loss for the input. The blue shaded region represents the ground truth for the input and the green line is the entropy of a small sliding window of the reconstruction loss. The lower graph shows a zoomed in picture of the reconstruction loss and ground truth.

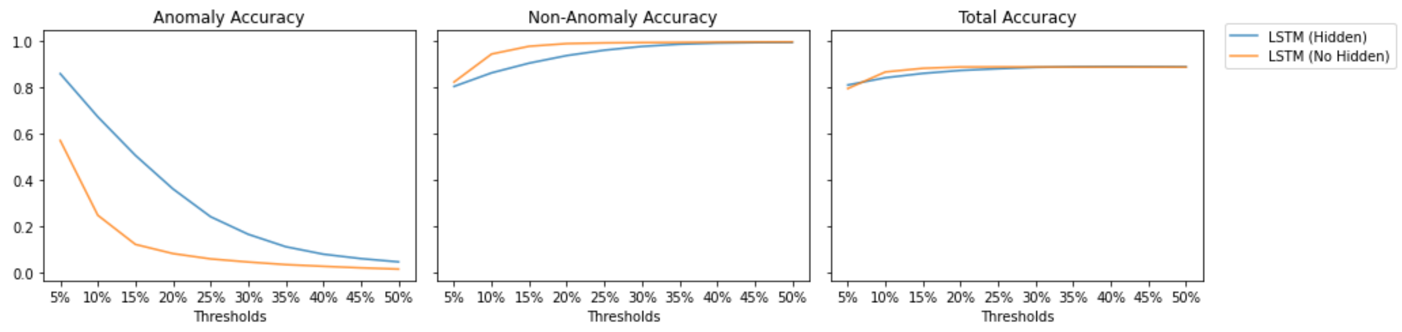

Fig 5. Comparison of different accuracies for AutoEncoder with using hidden states for decoder vs using output of the bottleneck LSTM layer forwarded to decoder.

plots the accuracies of both for different considerations of thresholds.

V. Semi-Supervised Suppression

In the previous section, we used normality modeling to train the AutoEncoder. From Fig 4. we see that the loss generated from reconstructed input forms a pattern around anomaly regions. We propose an extension to the AutoEncoder, which learns the patterns within the generated loss sequence to identify the spikes in the signal that correlate with the anomalies.

We further use a sliding window of the loss sequence to recognize these loss patterns, which is passed through a 1D convolution with 32 filters and three fully connected layers. Fig 6. shows the architecture of this extension. We use semi-supervised learning to try and suppress the false positives and false negatives to train this network. The loss sequences presented to the extension are taken from the AutoEncoder that uses LSTM hidden states from the bottleneck layer for the decoder.

We also note that the data imbalance plays a vital role in normality modeling, as seen from the output of the AutoEncoder. This imbalance is the basis of our anomaly detection; however, it causes problems when training with traditional classifications methods such as the one proposed for the extension. To counter the imbalance, we use a weighted mean squared loss. Before taking a mean over the losses of the training batch, we amplify the loss for anomaly samples by a factor of 10.

Fig 7. presents the extension with limited input and plot the results. With varied thresholds. The plotted accuracies show that the model can recognize anomalies very clearly with a semi-supervised factor of 50% for all threshold considerations. However, this comes at the cost of higher false positives. We, therefore, see a dip in both non-anomaly

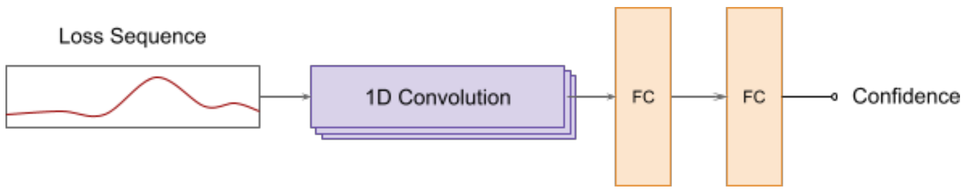

Fig 6. Extension to the AutoEncoder, a 1D convolutional layer with 2 fully connected layers, which take in the loss sequence generated from the AutoEncoder and output a confidence for the sliding window

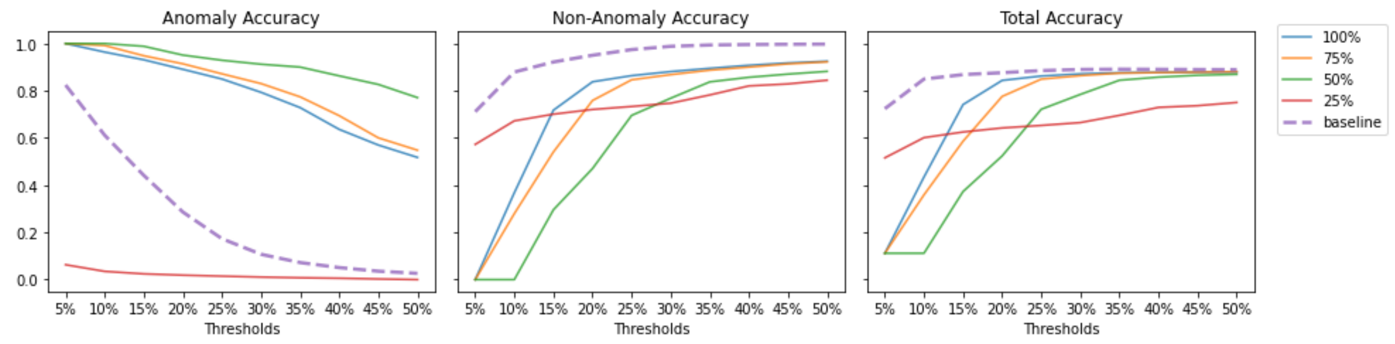

Fig 7. Comparison of accuracies with baseline results from AutoEncoder loss for Semi-Supervised learning with 25%, 50%, 75%, and 100% (completely supervised) of data

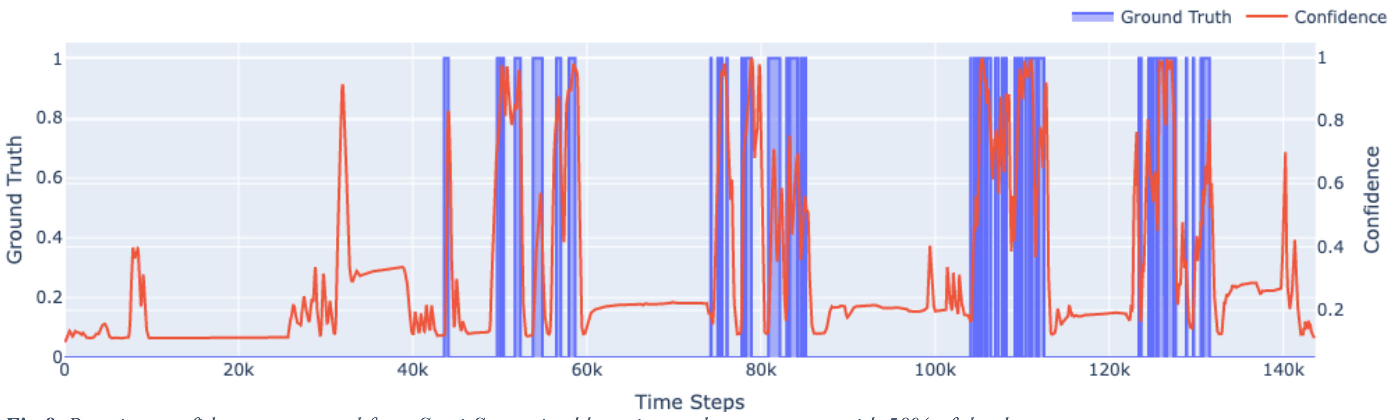

Fig 8. Running confidence generated from Semi-Supervised learning on loss sequence with 50% of the data.

accuracy and total accuracy. Regardless of the cost, our anomaly detection improves significantly and can be seen from the running confidence plot of a sample in Fig 8. Furthermore, as we decrease the amount of data that is presented to the extension, we see a decline in all accuracies to a point where, in the case of semi-supervised learning with very limited data (less than 50%), the model fails to learn at all.

VI. Limitations

We also note several limitations of our implementation that can be scope for further improvement:

1. **Testing:** The data comparisons are based on a range of testing thresholds, rather than a cross-validated threshold.

2. **Semi-Supervised Suppression:** To train the extension for false result suppression, we train a completely new neural network from scratch. It remains to be seen whether weight sharing can help improve the results, especially in cases of very limited data.

3. **Varied Results:** With different random seeds, even though the results captured the same improvement from unsupervised to semi-supervised training. The results for different thresholds varied significantly.

4. **More Experimentation:** It remains to be seen how the AutoEncoder performs under varied settings such as different sliding window sizes for input signals, a deep convolutional LSTM architecture for the AutoEncoder etc.

VII. Conclusion

In this paper, we proposed two hypotheses. (1) Given a binary classification problem with a heavily imbalanced dataset or anomaly detection, we can identify anomalies by learning and overfitting the normality in the data. (2) Using the signal from our normality modeling, we can further use semi-supervised learning to suppress the false positives and false negatives. While we can support (1) with the results from the AutoEncoder. Arguing for (2) proves to be a harder task. Regardless, of the robustness of semi-supervised learning, the model successfully learns a feature representation in an unsupervised fashion and is further able to improve on the anomaly accuracy with semi-supervised training.

References

[1] Bächlin, M., Plotnik, M., Roggen, D., Maidan, I., Hausdorff, J. M., Giladi, N., & Tröster, G. (2010). Wearable assistant for Parkinson's disease patients with the freezing of gait symptom. IEEE transactions on information technology in biomedicine : a publication of the IEEE Engineering in Medicine and Biology Society, 14(2), 436–446. https://doi.org/10.1109/TITB.2009.2036165

[2] Omar, S., Ngadi, M., Jebur, H., & Benqdara, S. (2013). Machine Learning Techniques for Anomaly Detection: An Overview. International Journal of Computer Applications, 79. http://dx.doi.org/10.5120/13715-1478

[3] Teng, M. (2010). Anomaly detection on time series. 2010 IEEE International Conference on Progress in Informatics and Computing, 1, 603–608. https://doi.org/10.1109/PIC.2010.5687485

[4] Ordóñez, F., & Roggen, D. (2016). Deep Convolutional and LSTM Recurrent Neural Networks for Multimodal Wearable Activity Recognition. Sensors, 16(1), 115. https://doi.org/10.3390/s16010115

[5] S. Fahn, "The freezing phenomenon in parkinsonism," Adv Neurol., vol. 67, pp. 53–63, 1995.

[6] N. Giladi, "Freezing of gait. Clinical overview." Advances in neurology, vol. 87, pp. 191–197, 2001.TITB-00037-2009

[7] M. Macht, Y. Kaussner, J. C. Mller, K. Stiasny-Kolster, K. M. Eggert, H.P. Krger, and H. Ellgring, "Predictors of freezing in parkinson's disease: A survey of 6,620 patients," Movement Disorders, vol. 22, no. 7, pp. 953–956, 2007.

Appendix

A. Model Summary

1) LSTM AutoEncoder

```
Encoder
=================================================================
                Kernel Shape     Output Shape   Params Mult-Adds
Layer
0_layers.LSTM_0            -  [1000, 24, 128]  71.168k   70.144k
1_h_activ                  -  [1000, 24, 128]        -         -
2_layers.LSTM_1            -   [1000, 24, 64]  49.664k   49.152k
3_h_activ                  -   [1000, 24, 64]        -         -
4_layers.LSTM_2            -    [1000, 24, 8]   2.368k    2.304k
5_out_activ                -    [1000, 24, 8]        -         -
-----------------------------------------------------------------
                      Totals
Total params          123.2k
Trainable params      123.2k
Non-trainable params     0.0
Mult-Adds             121.6k
=================================================================

Decoder
=================================================================
                Kernel Shape     Output Shape   Params Mult-Adds
Layer
0_layers.LSTM_0            -   [1000, 24, 64]  18.944k   18.432k
1_h_activ                  -   [1000, 24, 64]        -         -
2_layers.LSTM_1            -  [1000, 24, 128]  99.328k   98.304k
3_h_activ                  -  [1000, 24, 128]        -         -
4_layers.LSTM_2            -    [1000, 24, 9]   5.004k    4.932k
-----------------------------------------------------------------
                        Totals
Total params          123.276k
Trainable params      123.276k
Non-trainable params       0.0
Mult-Adds             121.668k
=================================================================
```

2) Semi-Supervised Network

```
=========================================================
        Kernel Shape    Output Shape  Params Mult-Adds
Layer
0_conv1  [1, 32, 10]  [500, 32, 491]   352.0   157.12k
1_pool             -   [500, 32, 49]       -         -
2_l1     [1568, 500]      [500, 500]  784.5k    784.0k
```

```
3_l2          [500, 50]        [500, 50]  25.05k     25.0k
4_l3            [50, 1]         [500, 1]    51.0      50.0
5_sigm                -         [500, 1]       -         -
---------------------------------------------------------
                        Totals
Total params          809.953k
Trainable params      809.953k
Non-trainable params       0.0
Mult-Adds              966.17k
=========================================================
```

B. *Training Configuration*

1) *LSTM AutoEncoder*

```
n_channels = 9 # number of sensor channels
len_seq = 24 # Sliding window length
stride = 1 # Sliding window step
num_epochs = 30 # Max no. of epochs to train for
num_batches= 20 # No. of training batches per epoch
batch_size = 1000 # Batch size
batchlen = 50 # No. of consecutive windows in a batch. If false, the
largest number of windows possible is used.
lr = 0.005 # Learning rate
```

2) *Semi-Supervised Network*

```
loss_len = 500 # Sliding window length
epochs = 15 # Number of training epochs
lr = 0.00005 # Sliding window length
batch_size = 1000 # Batch size
```