REPRESENTATION LEARNING FOR TIMESERIES USING AUTOENCODERS

Samy Tessier (s213459), Paul Seghers (s191675) Jonah Tabbal (s210158), Hector Devie (s212277)

https://github.com/TheJproject/deeplearning

1. INTRODUCTION

Today, the standard for road quality measurement revolves around standard visual inspection surveys, or standard automated inspection surveys both requiring expensive material such as specialized and heavily calibrated tools, as well as a time-consuming method of measurement. The aim of this project is to see to what extent data collected by ”naturally” driving a car along a road could be used to predict the quality of this road in an unsupervised way; with in mind the broader goal of cutting down the complexity and costs of estimating road quality. To this end we will be looking at time series corresponding to data collected by sensors mounted on vehicles as they drive on roads. Our task is to come up with a Deep Learning model which can learn a less-complex representation of these timeseries, and, if possible, use it to solve the binary ”good road / bad road ?” classification problem .

2. PREPROCESSING AND DATASET

The dataset was gathered using car sensor data from the Greenmobility fleet. The same data used and preprocessed in [1]. More specifically the data was constructed from two loops around Copenhagen, segmented into 100m . Each of the 3830 100m segment corresponds to a data point comprised of all attributes measured or informed about it: some are scalar (roughness, start point, etc), and some single-dimensional time-series (reconrding things like speed or acceleration along different axes).

The attribute we are trying to predict is the mean International Roughness Index (IRI, standard for the quality of a road) for a road segment as well as the vertical acceleration sustained by the vehicle for that specific road. It is an internationally agreed-upon unit typically within 2.0 and 5.0, with values higher than 4 being considered ”poor quality”: i.e. where reconstruction or intervention by the commune would be justified. The attribute that is commonly accepted to be best at predicting roughness of a road segments is the z-acceleration on this segments, i.e. how ”jaggedly” the vehicle bumps up and down.

Below is a geographical plot of the segments the vehicles drove on in copenhagen, colored by their (labelled) IRI mean, which was measured independently.

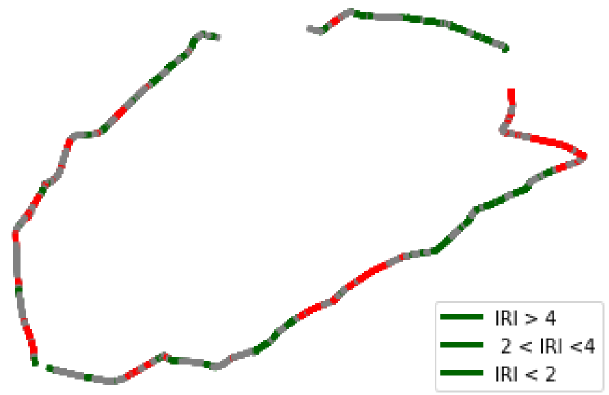

Fig. 1. Geographical plot of the segments colored by IRI mean.

We conducted a quick inspection on the dataset to see if there exist patterns in which the model can learn from. Figure 2 shows the the mean IRI against the mean difference in vertical acceleration for a series. The figure on the left is obtained using the absolute value of the mean and the one on the right using the mean difference between the values in the $0.85$and 0.15 quantiles. We see that the points roughly follow a linear trend, which gives us hope that training a model using these attributes could be interesting.

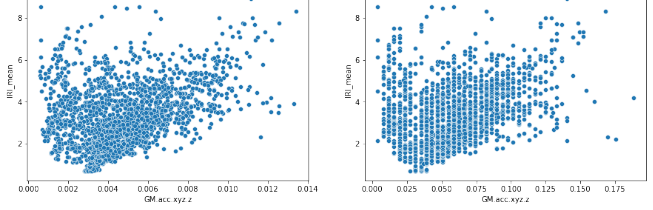

Fig. 2. Correlation between Δ-vertical acceleration vs IRI means on randomly sampled points. Right: by quantile

By sampling some series in different IRI grades, as seen in Figure 3. We see a significant difference in the z acceleration for these grade range. This is a definite sign that there is patterns to pick up for our model. At first glance, there seem to

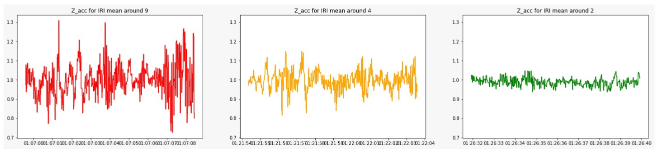

Fig. 3. Acceleration on segments where the mean IRI is: 5 (left), 4.0 (center), and 1.01 (right)

be properties that could remain when the timeseries are "compressed" into a point in a low-dimensional latent space, so the choice of using autoencoders for this task seems sound.

2.1. Preprocessing

Due to the way the measurements were made, the timeseries we have are of very different lengths: sometimes only a few hundred measurements are made on a 100m segment, while some segments have 50000.

Our sequences must be the same size to train an autoencoder on them. We tried two approaches:

- "Forced" method: truncating long series down to around 5000 points, dropping series with less that 1000 points, and interpolating series with between 1000 and 5000 points. We found values for this which left us with more than 85% of our 3830 observations, of which around a quarter had to be upsampled.

- the built in function `PackedSequence` from pytorch. It pads all sequences with a special encoded "empty" character, so that we avoid dimensionality problems in pytorch, but recurrent Neural Netowrk layers know not to update their weights when reading the padding.

We noticed when plotting them out that some timeseries had large "gaps", or constant intervals, but this wasn't too frequent and we decided to leave these in, as it might be linked to the behavior of the sensor used, or things that happen upstream of the Machine Learning tasks: this is supposed to be an unsupervised problem after all.
With the data properly formatted, it's time to get into the models.

3. MODEL AND RECONSTRUCTION

We used the L^2 norm on sequences as the reconstruction loss for all models. Once we had a working model, we found that changing the reconstruction loss used didn't affect performance much.
As a baseline for the reconstruction loss, we simply used We tried different approach to the AE model

- The first method we tried was using LSTM layers, as the `packedsequence` model doesn't allow convolutional layers. Small "easily trainable" -sized LSTMs (latent space of small dimension) gave very poor results.

- We used an AE using larger LSTM layers as used in [2]. We used a similar construction while using a different Dataloader. The model was quite slow with around 310 sec per epoch. The loss was going down but very slowly. We most likely would have needed around 150 epoch to have a decent result, which would have taken more than 12 hours to train.

- Finally, we ditched the packed sequence format, and moved on to Convolutional layers with dropout to gradually reduce dimension. We played around with the parameters of this model, and got best performance with the structure on Figure 4.

```
AE1(
  (encoder): Sequential(
    (0): Conv1d(1, 16, kernel_size=(7,), stride=(1,),...)
    (1): MaxPool1d(kernel_size=2, stride=2, ...)
    (2): ReLU()
    (3): Dropout(p=0.2, inplace=False)
    (4): Conv1d(16, 4, kernel_size=(7,), stride=(1,),...)
    (5): MaxPool1d(kernel_size=2, stride=2, ...)
    (6): ReLU()
  )
  (decoder): Sequential(
    (0): Conv1d(4, 16, kernel_size=(7,), stride=(1,),...)
    (1): ReLU()
    (2): Upsample(scale_factor=2.0, mode=nearest)
    (3): Dropout(p=0.2, inplace=False)
    (4): Conv1d(16, 16, kernel_size=(7,), stride=(1,),...)
    (5): ReLU()
    (6): Upsample(scale_factor=2.0, mode=nearest)
    (7): Conv1d(16, 1, kernel_size=(7,), stride=(1,),...)
  )
```

Fig. 4. Autoencoder Model Summary

The model we used is a mix of 1D Convolutional layers MaxPool1d, with some dropouts and Upsamples. We chose to use 1D Convolutional layers as they offer a clear interpretable visualization of what is happening for a timeseries. We use an Adam optimizer, we also tried a Stochastic Gradient Descent optimizer but it yielded similar results despite being slower, and we like having the same one as for LSTMs for comparison purposes (SGD is particularly slow on LSTM layers).

Compression size	Trivial MSE
2	0.000074
4	0.000183
8	0.000370
16	0.000610
32	0.000787

Table 1. Summary of MSE from compression size.

Depending on the compression size, i.e. the kernel size/scale factor of the AE , we get different trivial mean squared error (MSE).

We can see that with smaller bottleneck (i.e. less compression), we get better a better loss. We visualize the loss on Figure 3, where our model plateau at around 10^{-4} loss. The

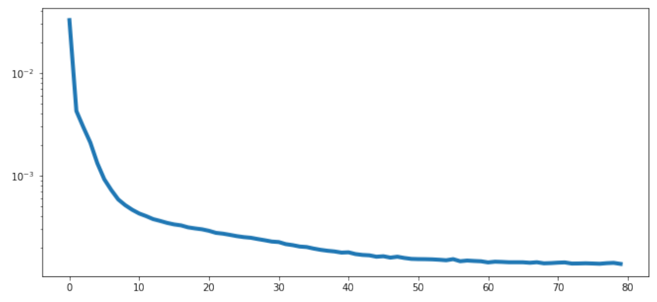

Fig. 5. Log plot of the reconstruction loss of our AE with the number of epochs, loss function used: L^2 (MSE)

loss in reconstructions greatly surpassed that of using naive MSE on the sequences, even up to "large" (10 to 16 times) compression of the longest sequences, it plateaus at around 0.000106.

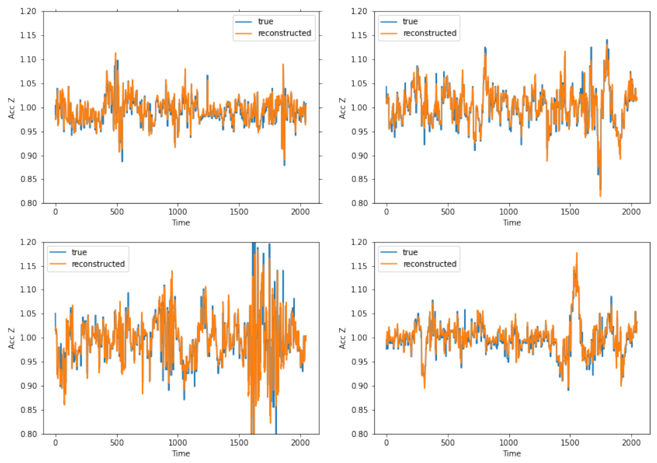

Fig. 6. comparison of sequences vs. their reconstruction through the autoencoder described below after training 80 epochs. This autoencoder

Since the reconstruction looks decent, we know that the latent space can be taken a reliable representation of the time-series, so the idea is now to push the problem further and see if there is any nice clustering inside this latent space.

4. LATENT SPACE VISUALIZATION, T-SNE, DBSCAN

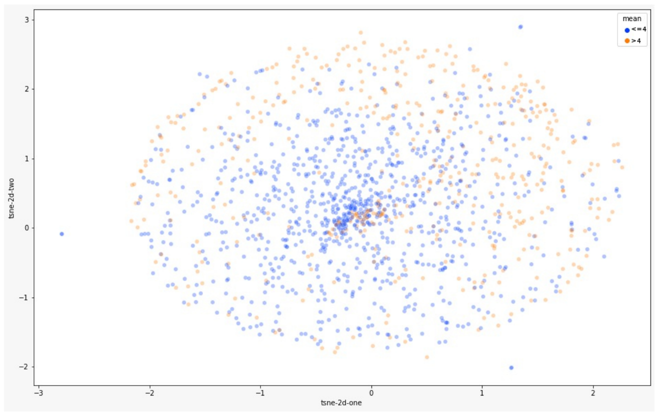

Fig. 7. Visualization of the t-SNE step on the latent space of our AE, colored by corresponding IRI-mean value

The trained model learn the 'structural similarities' between the time series. This similarities appear in the 'hidden' space, the latent space of the model. t-SNE (t-sensitive Stochastic Nearest Neighbors) is algorithm that works similarly to PCA to map a high dimensional space to a 2D or 3D one.

We see from Figure 7 "bad roads" seem to be clustering around one area of the representation: the top right. Moreover, the central dense cluster is overwhelmingly made up of "good" roads. In orange and blue it looks like the Firefox logo upside-down. Let's run DBSCAN on the output of the

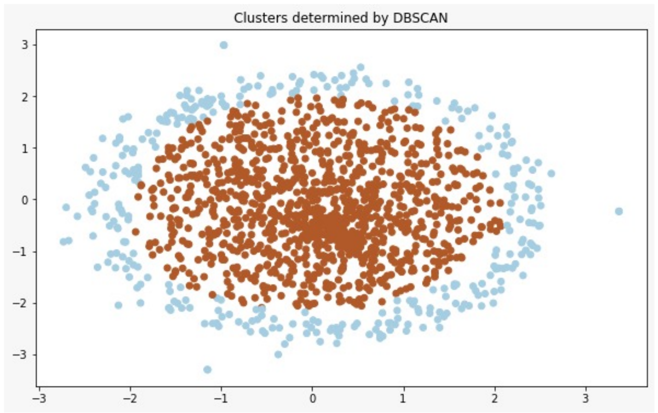

Fig. 8. DBSCAN with $\epsilon = 1.85$

te-SNE visualization to convince ourselves whether or not the "cluster" we see is significant, or if our minds played tricks on us because we really wanted there to be a cluster. DBSCAN (Density-based spatial clustering [3]) is appropriate here as t-SNE preserves the ratios between distances, and when we have a space where this is done and a binary classification problem, DBSCAN will (given a sensitivity as a hyperparameter) let us easily visualize the separation between the ”core” points and the ”outliers”: in our case what we would like to be two clusters. Unfortunately it seems our hopes end here, by binary searching for a value of ϵ which clearly separates outliers, we see that there isn't a significant separation between the clusters, as shown on Figure 8 :(

5. CLASSIFICATION

We would now like to solve the classification problem Given that we don't observe a clear clustering, it doesn't seem promising to use the latent space for classification and our hopes aren't high. we will try to see if the latent space of our. Here we compare the loss function between the following:

- simple Linear Regression (predict based on average Δ between points in the timeseries)
- LSTM classifier trained using the raw sequences (and their labels)
- LSTM trained on the latent space

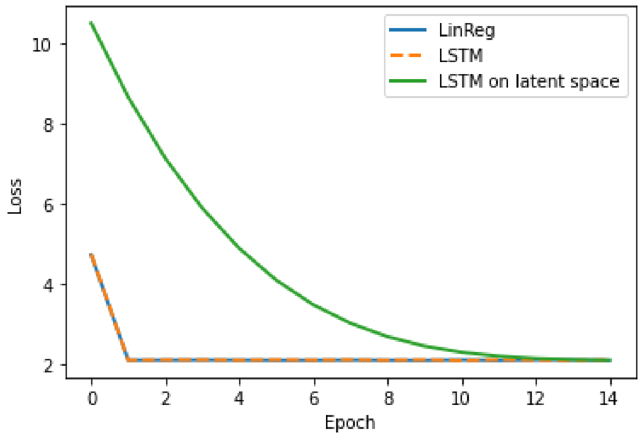

Fig. 9. Loss for classification

As we can see on 9, the results are once again slightly disappointing: with the same amount of training, our latent space does not really improve in performance over classification done with an LSTM which trains over the entire sequences (reconstruction loss bounded from below for both, training on latent converges slightly slower wrp. to epochs). Both are slow to train. However, even if it converges slowly ”mathematically” by a few powers (in log-plot, so linearly in the number of epochs), it trains considerably faster ($3\times$); this is due to the fact that training on the latent space means we can use LSTM layers with much smaller hidden layer sizes.
With this last observation in mind, let us take a step back and look again at the original problem of cutting humans out of road classification. From the dataset we have, and the discussions with we see that collecting data is not hard at all, but labelling it is. It is thus worthwhile to consider the case of seeing if a regression on the latent space with a lot of unlabelled data can beat conventional regression on a small number of data points. This makes sense, as unlabelled datapoints are magnitudes ”cheaper” (cost, time, effort) to obtain than labelled ones.

6. ”CHEAPER” CLASSIFICATION BY USING LATENT SPACE

We ran the same tests as in the previous sections (all hyperparameters, networks, optimizers, etc, kept the same), except this time we trained the LSTM classifier on a slice of `200` observations from our training data (the train/test split was circa 2000/1000).

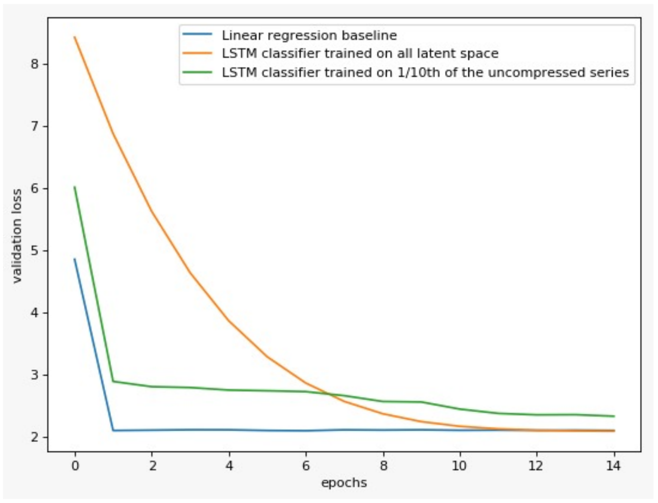

Fig. 10. Comparison of the performance of training on the latent space vs. on a small part of the regular dataset, for the binary classification problem

From this section, we can see one use we have found: since training a classifier network on the latent space is much faster than on the entire sequences which yielded these representations, it is reasonable to compare the performance of training on more latent space points than regular ones. And here, for 15 epochs we do see that a classifier trained on the latent space is somewhat better than a classifier trained on a fraction of the entire sequences; Though it will likely reach the same loss after more epochs.

7. CONCLUSION

We preprocessed the road roughness data to fit an autoencoder composed of 1D Convolutional layers. We tried different compression size for the latent space to get a good MSE. We then tried to visualize the latent space through t-SNE and DB-SCAN but it didn’t give very interesting results. However, we got good series reconstruction with low MSE. Finally, we observed that it is possible to leverage the speed boost of a classifier trained on the latent representation of timeseries. Given the fact that embedding a new observed timeseries into the latent space once the encoder is trained is rather cheap, this is an indication that such approaches could play a role in time-series classification tasks that arise from real-life problems, though clearly not with the models we managed to create for this project.

8. REFERENCES

[1] Milena Bajic, Shahrzad M. Pour, Asmus Skar, Matteo Pettinari, Eyal Levenberg, and Tommy Sonne Alstrøm, “Road roughness estimation using machine learning,” 2021.

[2] Venelin Valkov, “Time series anomaly detection using lstm autoencoders with pytorch in python,” 2020.

[3] Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu, “Dbscan revisited, revisited: Why and how you should (still) use dbscan,” *ACM Trans. Database Syst.*, vol. 42, no. 3, jul 2017.