Supporting Information for *"Low-Dimensional Embeddings for Interaction Design"*

Marius Rusu[1], Svenja Schött[1], and Roderick Murray-Smith[1]

[1]Affiliation not available

May 26, 2021

Abstract

This Supporting Information includes:

- in-depth information on the setup and implementation
- additional figures, tables and equations relevant to the article
- interactive code examples building on the shared data

Corresponding author Email: *Roderick.Murray-Smith@glasgow.ac.uk*

Setup and Implementation

Unity Application for Data Recording

We developed a Unity application with the help of the Dexmo Unity SDK that allows us to record synchronized data from both the Dexmo hand and the additional accelerometer at a stable framerate of 30 Hz. The Dexmo Unity SDK provides glove data at 200 Hz by transmitting packages from the glove to a python server and further to the Unity application in a blackbox fashion. For the additional accelerometer, we use a custom application that transmits acceleration and gyroscope values as TCP packages to the Unity application, where we synchronize the datastreams to 30 Hz. We have the option to manually set markers in the data by pressing keys, which allows us to mark the beginning and ending of certain actions such as gestures. The synchronized data including potential marker can be streamed to both a csv file and a python server. Streaming to csv files is helpful for recording and later analyzing the data as in section *Recording high-dimensional data*, while streaming to the python server allows us to evaluate data directly and couple in interaction tasks such as in section *Designing simple interaction.*

Record and process high-dimensional data

As mentioned in section *Recording high-dimensional data,* we recorded 20,000 samples of data for the training set as well as five repetitions of three general and three accelerometer specific gestures for future evaluation from four users each.

The data in its original state is comprised of the following features:

- 5 dimensions: Bending state of each finger ranging from 0 (fully stretched) to 1 (fully bent)
- 5 dimensions: Bending velocity of each finger derived from the difference between consecutive bending states and the time passed
- 18 dimensions: Acceleration and gyroscope values from three accelerometer devices placed on the back of the hand, the thumb and the index finger

To smooth the data, we applied a Savitzky–Golay filter with the window length 5 and polynomial order 2. Additionally, we scaled the data to the range from 0 to 1 using a MinMax scaler to obtain a uniform domain for the various features.

Train and optimize the autoencoder

As stated in section *Optimizing and training the autoencoder*, we trained three autoencoders that operate of different dimensionalities using the PyTorch framework and the GPyOpt library. To keep the different models as uniform as possible, all autoencoders consist of three layers of interchanged Linear and LeakyReLu modules that gradually reduce the dimensionality. As an optimization algorithm we use Adam in combination with the MSE (Mean Squared Error) loss function to minimise the distance between original and reconstructed features.

In addition to the MSE, we introduce the custom regulating factor MND (Mean Normalized Distance). MND aims at reducing the distance between successive samples in time in the embedding and should therefore smooth the trajectories of potential gestures in the two-dimensional latent space. Since the MND is only a regulating factor, we apply the weighting coefficient of 0.1 to its value before calculating the final loss L for the optimization algorithm as

$$L = MSE + \alpha MND$$

where the coefficient $\alpha = 0.1$. Figure 1 visualizes the impact of the custom regulator.

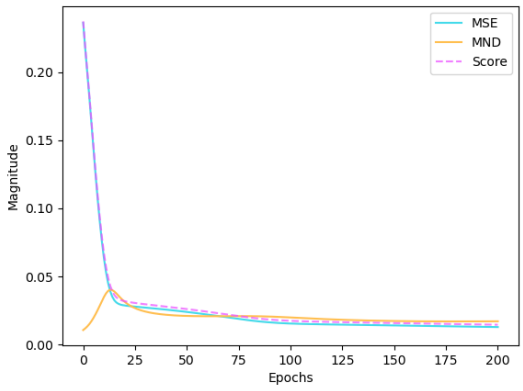

Figure 1: Learning curve represented by the MSE, MND and the final score during an explanatory training process. We can observe how the optimizer at first focuses on the MSE at the cost of the MND and then proceeds to also reduce the MND.

Furthermore, we performed a Bayesian hyperparameter optimization on the optimizing parameters learning rate and weight decay using the standard Gaussian Process model and Expected Improvement acquisition strategy. As optimization score, we used the equation mentioned above. While the different autoencoders

have the same structure, the Bayesian optimization allowed us to use individually optimized hyperparameters for the training process and thus improve the performance. Table 1 offers an overview of the exact hyperparameters.

Model	Dimensionality	Layers	Learning Rate	Weight Decay	MSE	MND
Model A	5 to 2	3	$2.1*10^{-5}$	$9.85*10^{-5}$	0.01	0.02
Model B	10 to 2	3	$9.68*10^{-6}$	$4.24*10^{-5}$	0.01	0.01
Model C	28 to 2	3	$6.02*10^{-6}$	$2.78*10^{-5}$	0.01	0.01

Table 1: Overview of the autoencoders and their characteristics. The optimization score of the autoencoders is comprised of the Mean Square Error (MSE) and Mean Normalized Distance (MND).

Interactive Code Examples

The major contribution of the article is the use of autoencoders to make high-dimensional movement data comprehensible for designers. We aim at bringing the machine-learning team and interaction design team together and build a common ground for discussion. For this reason, we want to share our code and data with the readers and hope to support future design processes.

The following figures contain Python notebooks that can be used to experiment with the provided data and autoencoder models. In case of any runtime issues with the current Python environment, we additionally offer access to our repository containing all code, data and models mentioned in the article. The repository also contains copies of the Python notebooks linked to the figures.

Visualizing two-dimensional embeddings and trajectories of gestures

This Python notebook allows us to visualize 15 different recorded gestures such as 'Screwing and unscrewing' or 'Drumming' in the low-dimensional space of *Embedding A*, *B* and *C*, which are discussed in the article. This gives us interesting insight into the nature of those every day movements.

Visualizing simplified glyph representations of two-dimensional embeddings

With this Python notebook we can generate the article's glyph visualizations of the three *Embeddings A*, *B* and *C*. This simplified plot gives us insight into the topology of the two-dimensional embedding space.

Training new autoencoder models

This Python notebook offers some of the functionality used to train our autoencoder models. Using this code, we can generate simple models with predefined hyperparameters of different input datasets. We recommend interested readers to access our repository for direct access to our hyperparameter optimizer. As this is an intensive process, we recommend to run the optimization on specialized hardware.

In case of any issues or questions about the Python notebooks, feel free to contact the corresponding author.

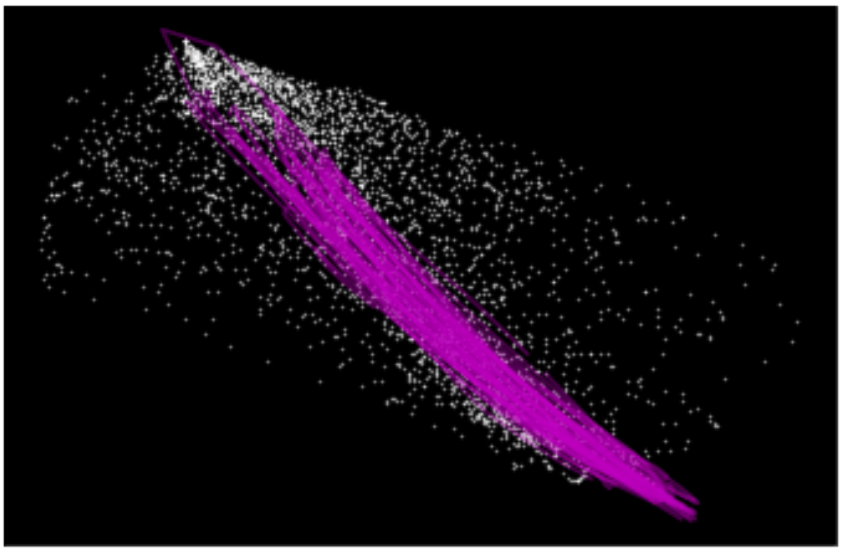

Figure 2: Embedding A containing the trajectories of the gesture 'Pinching'

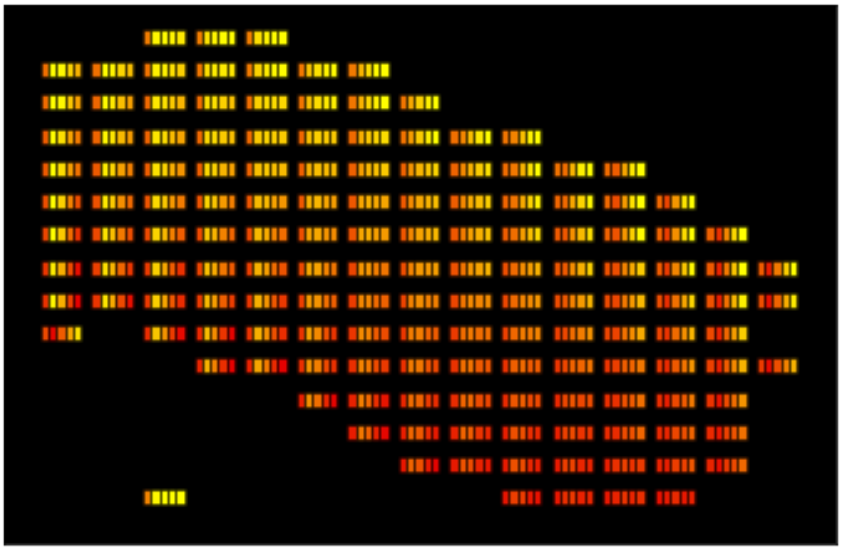

Figure 3: Simplified glyph visualization of Embedding A

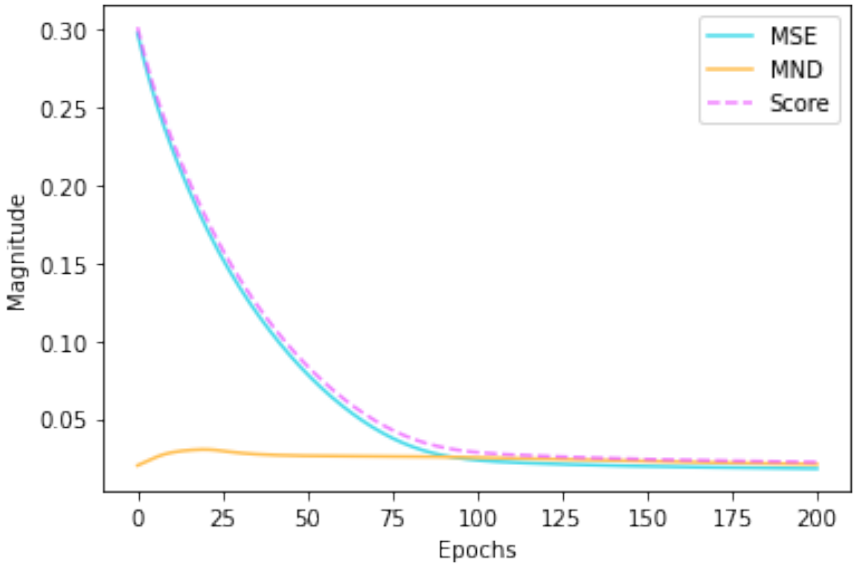

Figure 4: Learning curve of an exemplary training process