Underrated - Machine Perception Report

Yarden As
yardas@student.ethz.ch

Simon Bing
bings@student.ethz.ch

Florian Mahlknecht
fmahlknecht@student.ethz.ch

ABSTRACT

Human motion prediction has a wide range of applications, spanning from computer graphics to autonomous driving. To address this challenge, we implement an encoder-decoder based Recurrent Neural Network model to predict human motion from observations in the past. We explore a number of extensions to our base model, focusing on the effects of different representations of the data our model is trained with, as well as the pre-processing of this data.

1 INTRODUCTION

The prediction of the motion of others is central to many tasks that humans encounter on a daily basis. It allows us to efficiently interact with a large number of other humans without necessitating explicit communication. Think of walking across a crowded intersection or pushing along a shopping cart without bumping into others as simple examples. If we wish to build machines that are capable of such tasks, or even more complex ones, predicting human motion is something they must be capable of. The application space for this task is large: computer graphics, virtual and augmented reality or autonomous driving are prime examples of where human motion prediction is in demand.

The large feature space, many degrees of freedom and complex spatio-temporal relations are what make this task as challenging as it is. Additionally the complex dependency of context and intent of humans moving around in the world make predicting human motion difficult for classical methods. This in part has given rise to a number of deep learning approaches, but the task is still far from being solved.

We focus on implementing a machine learning based model to address the task of reliably predicting the motion of a human from observations of the motion sequence in the past. The model we have opted to implement for this is an encoder-decoder based Recurrent Neural Network. While our primary focus lies on achieving the best possible performance in the task at hand, we also investigate multiple extensions to our base model. We focus on the influence that different underlying representations of the data we train our model with have, in conjunction with various pre-processing of the data in the respective representation domain.

2 METHODOLOGY

We base our model on state of the art research [1, 2, 4] and focus our contribution on testing different normalization approaches for these quite uncommon data structures, which encode rotations.

The experiment runs are evaluated using the mean joint angle differences between all joints, summed over all target prediction frames. Furthermore, we evaluate qualitatively how natural the models predict the continuation of human body motions by generating for all test runs the respective video outputs.

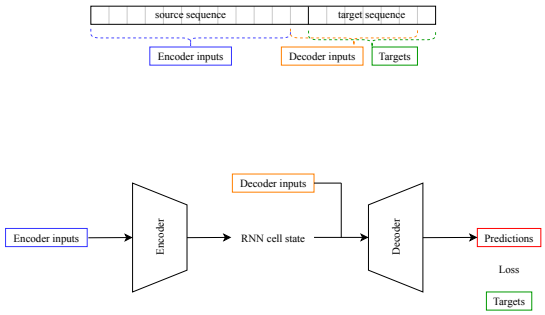

Figure 1: A diagram of the sequence-to-sequence model: the encoder embeds the seed sequence into latent representations, while the decoder uses these representations to kick-off its own predictions.

3 PREDICTION MODEL

In its raw form, each data sample is represented as 144 frames, each holds the values of 3D rotation matrices describing the configuration of the 15 joints of a humanoid model. As explained before, since modelling correctly this form of data requires capturing the complex spatio-temporal relations of it, we decided to use a Recurrent Neural Network (RNN) based sequence-to-sequence model [4].

Model architecture. We construct a sequence-to-sequence model that predicts a sequence of 24 frames, conditioned on a seed sequence of 120 frames. Our sequence-to-sequence model uses two RNNs in an encoder-decoder architecture. The encoder takes as an input the seed sequence and embeds it into a hidden state vector that holds latent information of the human's motion. We discard the outputs of the encoder's RNN and only use its hidden state vector. The hidden state vector is then passed over to the decoder for prediction. At inference, we first "warm-up" the decoder with the encoder's hidden state and the last frame of the seed sequence. Following that, at each prediction step we use the decoder's RNN output and hidden state of the previous prediction frame to predict the next frame. Figure 1 gives an overview of the sequence-to-sequence architecture.

Training. In [4] and as seen in Figure 1, at training, the decoder's input at each time step is the training target of the previous time step. This type of training procedure is called "Teacher Forcing" [9]. Although this training method is currently the most common one, one failure mode of it happens when the decoder's predictions at inference time start to diverge from the training data distribution. In these situations, the decoder is not robust enough to recover from its own mistakes and the output sequence might be heavily biased. This discrepancy between training and inference can be reduced by using the decoder's predictions at training time the same way as we use in inference time. Although this approach can potentially make

the model more robust, it might slow down training and lower overall performance since at the start of training the decoder's outputs are not well tuned. [2] proposes an approach that interpolates between these two extremes as training progresses. More specifically, it alternates between using the decoder's own predictions and Teacher Forcing by sampling from the Bernoulli distribution $\text{Ber}(p)$. Whereby, without loss of generality, p is the probability of using Teacher Forcing for the next prediction. To interpolate between the two approaches, we decay p linearly throughout training. For example, if we fix $p = 1$ our model will be trained only with Teacher Forcing. Finally, we minimize the Mean Square Error (MSE) loss between our decoder's predictions and targets of the corresponding frame.

Extensions to the sequence-to-sequence architecture. Our main extension to the base sequence-to-sequence model is by using a Structured Prediction Layer [1, SPL] to incorporate prior knowledge of our problem to the model. More specifically, the SPL assembles the kinematic chain of a humanoid model by using a small neural network per joint. Furthermore, we use the following tricks to improve learning stability and model robustness:

- Train our model to predict the residuals to the next frames as described in [4].
- Use dropout on the encoder's hidden state to help making the model more robust as suggested in [3].
- Share weights between the encoder and the decoder, as suggested in [4].

4 NORMALIZATION AND DATA REPRESENTATION

A recurring assumption behind many design choices in machine learning is to have independent and identically distributed Gaussian input features. The usual approach is to standardize feature by feature, in order to have zero mean unit variance data, with values all in the same order of magnitude.

The raw data in our case are rotation matrices in $SO(3)$. As 3x3 matrices they contain 9 values, however they can not be independently chosen. In fact, there are 6 constraints reducing the degrees of freedom to 3, as one would expect for a rotation in 3D space. Expressed through the columns $\mathbf{u}_i$ of a rotation matrix, the constraints are given by:

(1) $\|\mathbf{u}_i\| = 1$ for $i = 1, 2, 3$
(2) $\mathbf{u}_3 = \mathbf{u}_1 \times \mathbf{u}_2$

We can therefore conclude, that entries of a rotation matrix are not independent, as the rotation matrix representation incorporates redundancies by design. The most straight forward approach of standardizing the data element-wise is therefore not rigorously correct.

Similar to [8] which uses quaternions to encode rotations, we decided to try out different representations and ways to standardize the data in a more meaningful way.

4.1 Axis Angle

An intuitive representation of rotations in 3D space is the axis angle format, which consists of a rotation axis direction, given as a unit vector $\hat{\mathbf{u}}$ and an angle θ. Through multiplication we can incorporate the information in a single vector, the so-called rotation vector:

$$\mathbf{v} = \theta\,\hat{\mathbf{u}} \tag{1}$$

representing a 3D rotation with just 3 elements, i.e. without any redundancies. The remaining issue with this representation is the ill-defined direction for a 0° rotation, where the matrix representation simply falls back to the identity mapping.

To convert between axis angle and matrix representation, the Rodriguez formula can be used, see [5]. During implementation we made use of the tensorflow graphics library, which provides the respective functions already.

The scalar rotation angle can be used to get some insights into the training dataset.

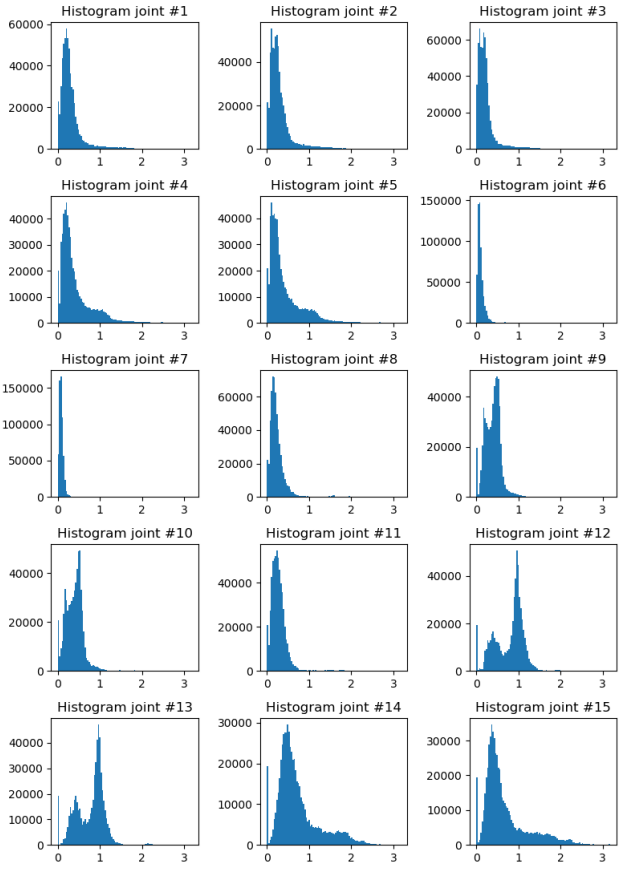

Figure 2: Joint rotation histograms in [rad]

As figure 2 illustrates, the joints shows various distributions, some of them multi-modal, some of them with relatively high variance and others very sharp with low variance.

4.2 Log space transform for rotation angle

Most of the distributions show a one-sided Gaussian with a longer right tail for higher rotations. The log space which increases the resolutions in the lower regions and decreases the resolutions for

higher values in a logarithmic fashion, could reshape those distributions into proper Gaussians. The results of this idea are shown in figure 3.

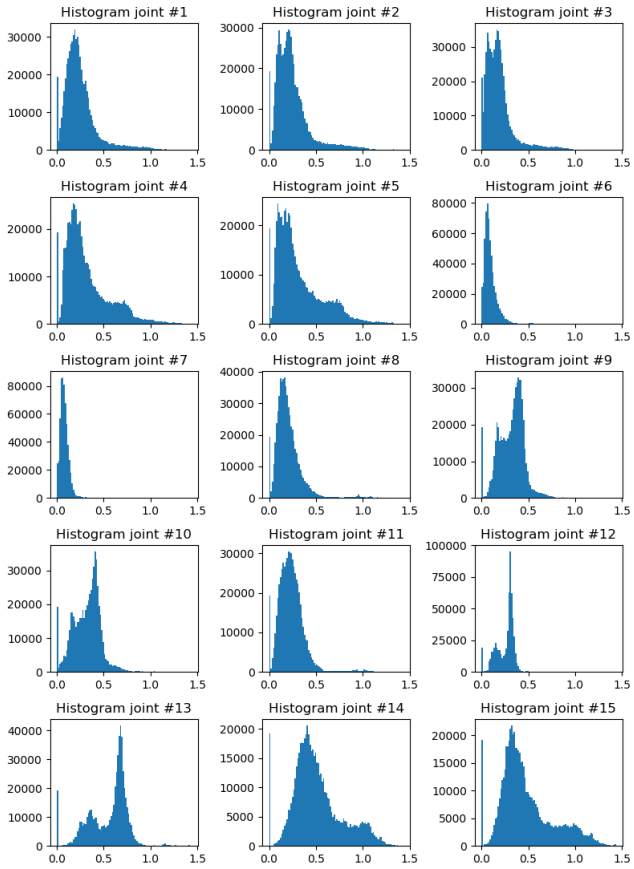

Figure 3: Joint rotation histograms in [log]

Indeed, qualitatively the distributions in the log space are more similar to the Gaussian bell curve.

4.3 Fréchet Mean

In the context of data standardization, computing the statistics of our data-set is not as straightforward as most regression tasks. Moreover, even if we transform our data to axis-angle representation by using the Rodriguez formula [5], the arithmetic statistics (i.e., mean and standard deviation computed arithmetically) do not represent their true meaning. For example, computing the arithmetic mean in axis-angle representation does not necessarily serve as a good measure of the data-set's central tendency. To better capture the mean rotation, one has to solve

$$R_m = \arg\min_{R \in \mathrm{SO}(3)} \sum_{i=1}^{N} d^2(R, R_i) \tag{2}$$

where $d(\cdot,\cdot)$ is the geodesic distance in SO(3) and is given by

$$d(R_1, R_2) = \|R_1 - R_2\|_F = \mathrm{tr}(R_1^\top R_2). \tag{3}$$

Finally, $\|\cdot\|_F$ is the Frobenius norm. By solving equation (2), we find the rotation matrix that minimizes the distance to all other rotations in the data-set, thus, we find the rotation that represents the central tendency of the data-set [7]. Practically, to solve equation (2), we use the `geomstats` library [6].

4.4 Inverse rotation matrix

When standardizing scalar data, the approach is straight-forward: given the empirical mean μ and standard deviation σ the data x is standardized by means of applying the formula $\hat{x} = \frac{x-\mu}{\sigma}$.

Motivated by this simple formal notation, we aimed to transfer this idea to rotation matrices, since our data is presented to us in this representation. As we pondered on how to implement this idea in the domain of rotations, two primary questions arose: how are statistics such as the mean and standard deviation of rotations calculated and which operations in the domain of SO(3) matrices correspond to subtraction and division?

Simply taking the element-wise mean and standard deviation is not the solution, as the resulting matrices are no longer guaranteed to be valid rotations. After research in the literature, we found that mean rotations could be calculated using the Fréchet mean in SO(3), as described in Section 4.3. To the best of our knowledge, no such concept exists that could be used as a proxy for the standard deviation in the formalization for the scalar case. Given this missing component, we opted to simply apply the inverse of the calculated mean rotation to the data. We interpret this operation as the analog to subtracting the empirical mean in the scalar case. The mean is calculated per joint and the standardization is then also applied for each joint separately.

Although the scaling by the standard deviation is missing in this setting, we argue that this is less of an issue than one might initially assume. Rotations matrices are by construction numerically all in the same order of magnitude, and since we ensure that our "mean-subtraction" operation returns valid rotations matrices, this conditioning is preserved and no further scaling is needed. Examples of the mean rotation matrix found per joint can be seen in Figure 4.

5 EXPERIMENTAL RESULTS

We conduct a number of experiments with different data representations in combination with various normalization schemes. Other than the mentioned representations and normalizations, the tested models have identical hyperparameters and are trained for the same number of epochs, to provide comparability between the approaches. Results of the experiments are illustrated in Table 1 and Figure 5.

Not standardizing the data and keeping it in its standard representation yields the best results. Among the tested standardization schemes, normalizing element-wise by the Fréchet mean performs best. The axis angle representation seems to diminish prediction quality, especially in combination with a log transform of the data. From the training curves in Figure 5 it is also evident, that the margin between the Fréchet and basic standardization is very small, suggesting that more rigorous comparison of these approaches may be necessary to fully conclude on the one or the other's dominance.

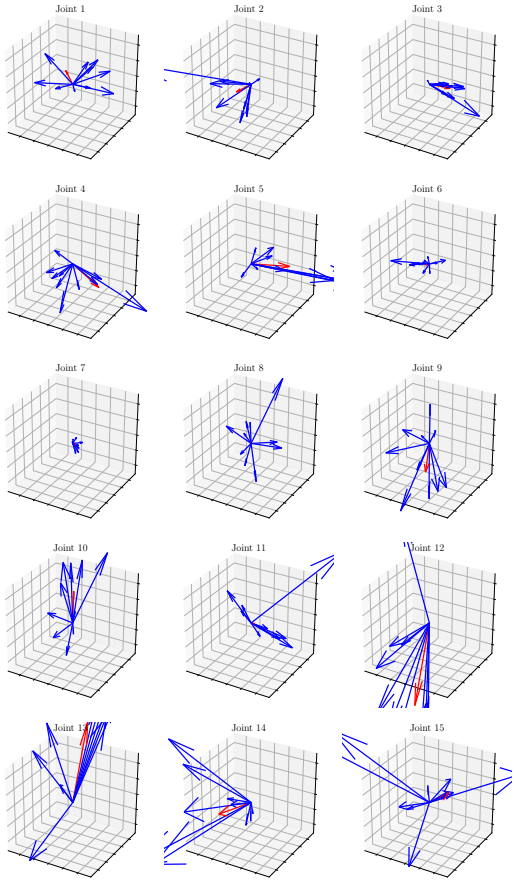

Figure 4: Visualization of mean rotations of a given seed sequence. The red arrows represent the mean rotation of the blue arrows, calculated per joint.

Experiment ID	Standardization	Data Representation	Score
<030>	basic	raw_data	**3.15**
<031>	basic	axis_angle	3.27
<032>	basic	axis_angle_4	3.26
<034>	log	axis_angle_4	3.32
<035>	log	axis_angle	7.31
<036>	frechet	raw_data	3.18
<037>	frechet	axis_angle	3.40
<038>	inv_rotmats	raw_data	3.19

Table 1: Normalization approaches comparison

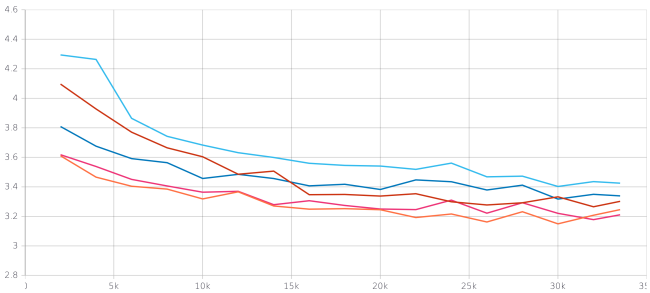

Figure 5: Validation metrics comparison [<037> light blue, <031> red, <034> dark blue, <036> pink, <030> orange]

6 CONCLUSIONS

In this project, we develop a sequence-to-sequence model and train it to predict human motion. Furthermore, we study the efficacy of different data representation and standardization schemes.

We observe no improvement with any of the tested data representations and standardization schemes, from which we conclude that the most effective way to improve performance in this task is by constructing more powerful architectures. Furthermore, we argue that choosing the right set of hyperparameters might have a stronger effect on the model's performance, even if we use problem specific notions such as the Fréchet mean. Nevertheless, we contend there is more room for rigorous research in this direction since it introduces the model more prior knowledge thus it should, at least theoretically, improve results.

REFERENCES

[1] Emre Aksan, Manuel Kaufmann, and Otmar Hilliges. 2019. Structured Prediction Helps 3D Human Motion Modelling. arXiv:cs.CV/1910.09070
[2] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. 2015. Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks. arXiv:cs.LG/1506.03099
[3] Partha Ghosh, Jie Song, Emre Aksan, and Otmar Hilliges. 2017. Learning Human Motion Models for Long-term Predictions. arXiv:cs.CV/1704.02827
[4] Julieta Martinez, Michael J. Black, and Javier Romero. 2017. On human motion prediction using recurrent neural networks. arXiv:cs.CV/1705.02445
[5] Johan Ernest Mebius. 2007. Derivation of the Euler-Rodrigues formula for three-dimensional rotations from the general formula for four-dimensional rotations. arXiv:math.GM/math/0701759
[6] Nina Miolane, Alice Le Brigant, Johan Mathe, Benjamin Hou, Nicolas Guigui, Yann Thanwerdas, Stefan Heyder, Olivier Peltre, Niklas Koep, Hadi Zaatiti, Hatem Hajri, Yann Cabanes, Thomas Gerald, Paul Chauchat, Christian Shewmake, Bernhard Kainz, Claire Donnat, Susan Holmes, and Xavier Pennec. 2020. Geomstats: A Python Package for Riemannian Geometry in Machine Learning. arXiv:cs.LG/2004.04667
[7] Frank Nielsen and Rajendra Bhatia. 2012. *Matrix Information Geometry*. Springer Publishing Company, Incorporated.
[8] Dario Pavllo, Christoph Feichtenhofer, Michael Auli, and David Grangier. 2020. Modeling Human Motion with Quaternion-Based Neural Networks. *International Journal of Computer Vision* 128, 4 (April 2020), 855–872. https://doi.org/10.1007/s11263-019-01245-6
[9] R. J. Williams and D. Zipser. 1989. A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. *Neural Computation* 1, 2 (1989), 270–280.