Human Motion Prediction

Machine Perception Spring 2019
ETH Zürich

Metod Jazbec
mjazbec@student.ethz.ch

Rok Šikonja
rsikonja@student.ethz.ch

ABSTRACT

In this paper we explore modeling human motion prediction using sequence-to-sequence models. We do a review of relevant articles, propose two different architectures obtaining best scores and share our findings on different model hyperparameters and training.

This paper and corresponding project is part of the Machine Perception course in Spring semester 2019 at ETH Zürich.

1 INTRODUCTION

With raising demand, prediction of human motion has found several applications in many technological areas, such as robotics, human-computer interaction and autonomous systems.

Traditional predictive models have typically incorporate complex expert knowledge in the form of Markovian assumptions, smoothness or low-dimensional embeddings. Latest breakthroughs in deep learning and recurrent neural network (RNN) architectures have resulted in outperforming previous state-of-the-art model in many problems dealing with sequential data. Such methods have shown good performance also on human motion prediction task. The authors of [1] propose a Encoder-Recurrent-Decoder (**ERD**) network, which uses curriculum learning to jointly learn pose data representation and temporal dynamics. Some approaches include 3 layer LSTM recurrent neural network with a dropout autoencoder (**3LR-LSTM DAE**) to model temporal and spatial structures and combat error accumulation. [2] Current state-of-the-art approaches formulate the human motion prediction task as a sequence-to-sequence (**S2S**) problem, which is then modeled by RNNs in the encoder-decoder networks. *Martinez et al.* [4] show, that near state-of-the-art performance can be achieved by a simple zero-velocity model (**ZERO**), which does not attempt to model motion at all. Additionally, they introduce residual connections (**RES**), which model first-order motion derivatives, and introduce sampling loss (**SL**), where network is given its own predictions, instead of ground truth poses, as seeds in the decoder part of the network during training.

However, previously mentioned approaches suffer from error accumulation and severe motion discontinuity between ground truth input and predicted output sequences. In this view, the authors of [3] introduce geometry-aware loss, geodesic loss, which in contrast to Euclidean loss more naturally quantifies the similarity between two rotation matrices. Moreover, they propose two global recurrent discriminators, fidelity and continuity, which are trained in Generative Adversarial Network fashion. Fidelity discriminator (**FID**) distinguishes between short sequences, while continuity discriminator (**CON**) distinguishes between long sequences, thus forcing smoothness of the prediction and coherence between predicted and the input sequence.

1.1 PROBLEM FORMULATION

Given a sequence of pre-recorded human motion data, task is to predict, how will human motion evolve over a short period of time, or more precisely, for 24 frames or 400 milliseconds in the near future. For the sake of this task, training, validation and test datasets are provided. We train our model on the training set, where we take 120 frames as input sequence (2000 milliseconds) and try to predict next 24 frames. Model performance is evaluated on the validation set. Angular distance is used as an evaluation metric, which is defined as the angle, that is required to rotate predicted to the ground truth rotation matrix. Test set is used to report model performance for the sake of grading by our instructors.

Furthermore, for this task human motion is represented as a sequence of skeletal representations, i.e. we represent the human body as a set of its major joints which is organized in a tree-like, hierarchical chain that defines how the joints are linked together. Skeleton consists of 15 major joints, each joint is represented as a rotation relative to its parent in hierarchy tree as a rotation matrix $\mathbf{R} \in \mathbb{R}^{3\times3}$. Thus, every skeleton is altogether represented as a $15 \cdot (3 \cdot 3) = 135$ dimensional vector.

2 OUR METHOD

In the following sections we describe our model details and its parts, data representation, metrics used and training details.

2.1 ZERO VELOCITY MODEL

After careful literature review, we decided to first implement a simple zero-velocity model (**ZERO**), which will serve as a baseline model in assessing the performance of more complex models. Given an input sequence zero-velocity model predicts the last frame for every prediction step. This model does not contain any trainable parameters, therefore no training is required.

2.2 DATA REPRESENTATION

As described previously, the dataset contains human body representation in form of rotation matrices, which were then pre-processed into angle-axis representations. During training we used both and analysed the impact of data representation. In the evaluation and testing phase predicted rotation matrices and angle-axis representations are first transformed to valid rotation matrices. The latter have an advantageous property, since transformation from angle-axis to rotation matrix, always outputs a valid rotation matrix, whereas not every predicted rotation matrix is valid.

2.3 DATA STANDARDIZATION

In order to speed up training and increase robustness and stability data representations were standardized before training and de-standardized after prediction in evaluation or testing phase. If we

denote $\mathbf{v} \in \mathbb{R}^{135}$ the flattened rotation matrix representation for one skeleton, i-th component was standardized as:

$$\mathbf{v}_i = \frac{\mathbf{v}_i - \mu_{\mathbf{v}_i}}{\sigma_{\mathbf{v}_i}},$$

where $\mu_{\mathbf{v}_i}$ and $\sigma_{\mathbf{v}_i}$ denote the mean and standard deviation of i-th component computed on the entire training set. Same procedure was taken with angle-axis representations.

2.4 SEQUENCE-TO-SEQUENCE MODEL

As described in the literature, we have decided to reformulate the prediction task as a sequence-to-sequence (**S2S**) problem, namely input sequence to output sequence. In this regard we use many-to-many RNN architecture, where the encoder encodes the information of the input sequence in a RNN hidden state representation, which is then passed to the decoder, which at every time step t outputs a prediction for time step $t + 1$.

Let's define some quantities to mathematically describe our model and its parts. Let $\mathbf{X} = \{\mathbf{X}_1, \ldots, \mathbf{X}_{144}\}$ be a sequence of frames, where $\mathbf{X}_i \in \mathbb{R}^{135}$ represents i-th frame human body pose representation, among which frames $\mathbf{X}^e = \{\mathbf{X}_1, \ldots, \mathbf{X}_{119}\}$ are feed as a seed to the encoder. Sequence of frames $\mathbf{X}^d = \{X_{120}, \ldots, \mathbf{X}_{143}\}$ is feed to the decoder, whereas the set $\mathbf{X}^t = \{\mathbf{X}_{121}, \ldots, \mathbf{X}_{144}\}$ is used as a set of target poses for learning. Let's also denote an input sequence, known in evaluation and testing phase, $\mathbf{X}^i = \{\mathbf{X}_1, \ldots, \mathbf{X}_{120}\}$. Decoder outputs $\hat{\mathbf{Y}} = \{\hat{\mathbf{Y}}_{121}, \ldots, \hat{\mathbf{Y}}_{144}\}$, where $\hat{\mathbf{Y}}_i \in \mathbb{R}^{N_{cell}}$, are first propagated through a dense output layer to obtain predicted poses, i.e. $\hat{\mathbf{X}} = g^o(\mathbf{W}^o\hat{\mathbf{Y}} + \mathbf{b}^o) = \{\hat{\mathbf{X}}_{121}, \ldots, \hat{\mathbf{X}}_{144}\}$.

A dense layer is added before feeding $\mathbf{X}$ to the RNN, namely $\mathbf{Y}^r = g^i(\mathbf{W}^e\mathbf{X}^e + \mathbf{b}^e)$ and $\mathbf{Y}^d = g^i(\mathbf{W}^d\mathbf{X}^d + \mathbf{b}^d)$, with $\mathbf{Y}^e_i, \mathbf{Y}^d_i \in \mathbb{R}^{N_{in}}$. Additionally, when using sampling loss (**SL**) during training, previous step prediction $\hat{\mathbf{X}}_t$ is used to predict $\hat{\mathbf{X}}_{t+1}$ (instead of feeding ground truth $\mathbf{X}_t$ to decoder at every time step t).

In the following sections, module parts are introduced and described, which can be enabled at will by flag selection. In the section 3, we analyse their impact on the model performance.

2.5 RESIDUAL CONNECTIONS

Residual connections (**RES**) are added after the output dense layer. In this setting, RNN models first order derivatives, therefore, we obtain differences $\Delta\hat{\mathbf{X}} = g^o(\mathbf{W}^o\hat{\mathbf{Y}} + \mathbf{b}^o) = \{\Delta\hat{\mathbf{X}}_{121}, \ldots, \Delta\hat{\mathbf{X}}_{144}\}$. Finally, prediction at time step t is obtained by adding ground truth $\mathbf{X}_t$, or in case of **SL** $\hat{\mathbf{X}}_t$, to the modeled derivative $\Delta\hat{\mathbf{X}}_t$, i.e. $\hat{\mathbf{X}}_{t+1} = \mathbf{X}_t + \Delta\hat{\mathbf{X}}_t$.

2.6 DISCRIMINATORS

In order to promote smoothness and coherence with the input sequence, we followed GAN approach of [3]. Fidelity discriminator distinguishes between ground truth $\mathbf{X}^t$ and predicted $\hat{\mathbf{X}}$ output sequences, whereas continuity discriminator distinguishes between ground truth X and predicted $\{\mathbf{X}^i, \hat{\mathbf{X}}\}$ sequence.

Both sequences are first propagated through a dense input layer, parametrized by $\mathbf{W}^f, \mathbf{b}^f$ for fidelity and $\mathbf{W}^c, \mathbf{b}^c$ for continuity discriminator, with activation function g^i, and then passed through an RNN, where the last hidden states $h^f, h^c \in \mathbb{R}^{N_{disc}}$ are finally passed through a dense layer with sigmoid activation to get the probability of the sequence being real. Ideally, our predictor will learn to predict sequences, that discriminators would classify as real. If we denote $\mathcal{P}$ predictor, $\mathcal{D}_f$ fidelity and $\mathcal{D}_c$ continuity discriminator parameters, then we jointly optimize the following minimax objective:

$$\mathcal{P}^* = \arg\min_{\mathcal{P}} \max_{\mathcal{D}_f,\, \mathcal{D}_c} \lambda \cdot \left(\mathcal{L}^f_{\text{adv}}(\mathcal{P}, \mathcal{D}_f) + \mathcal{L}^c_{\text{adv}}(\mathcal{P}, \mathcal{D}_c)\right) + \mathcal{L}_{\text{pred}}(\mathcal{P}), \tag{1}$$

where λ is a loss-weighting hyperparameter and $\mathcal{L}_{\text{pred}}$, $\mathcal{L}^f_{\text{adv}}$ and $\mathcal{L}^c_{\text{adv}}$ denote losses to be minimized with $\hat{\mathbf{X}} = \mathcal{P}(\mathbf{X}^i)$. Losses are defined as follows:

$$\mathcal{L}_{\text{pred}}(\mathcal{P}) = \sum_{i=121}^{144} \sum_{n=1}^{15} \mathrm{d}(\hat{\mathbf{R}}^{(i)}_n, \mathbf{R}^{(i)}_n),$$

$$\mathcal{L}^f_{\text{adv}}(\mathcal{P}, \mathcal{D}_f) = \mathbb{E}_{\mathbf{X}^t}\left[\log(\mathcal{D}_f(\mathbf{X}^t))\right] + \mathbb{E}_{\mathbf{X}^i}\left[\log(\mathcal{D}_f(1 - \hat{\mathbf{X}}))\right] \text{ and}$$

$$\mathcal{L}^c_{\text{adv}}(\mathcal{P}, \mathcal{D}_c) = \mathbb{E}_{\mathbf{X}}\left[\log(\mathcal{D}_c(\mathbf{X}))\right] + \mathbb{E}_{\mathbf{X}^i}\left[\log(1 - \mathcal{D}_c(\{\mathbf{X}^i, \hat{\mathbf{X}}\}))\right].$$

2.7 ANGLE-AXIS REPRESENTATION AND GEODESIC LOSS

Given that human skeleton is represented as a set of rotation matrices for every major joint, our interest is to design a distance metric between two rotation matrices, which would in contrast to standard Euclidean distance incorporate information on the structure of 3D rotations.

3D rotation matrices form a Special Orthogonal Group SO(3) of orthogonal matrices with determinant equal to 1. SO(3) is a Lie Group, which is an algebraic group with a Riemannian manifold structure. In this view, our distance metric should quantify the shortest path between two rotation matrices on the manifold, which is not the case of the Euclidean distance. [3]

In order to define geodesic loss, we have to note, that a rotation can be expressed in many forms, here we will consider rotation matrix, $\mathbf{R} \in \mathbb{R}^{3\times3}$, and angle-axis, $\mathbf{a} \in \mathbb{R}^3$, representations. Angle-axis representation is defined as $\mathbf{a} = [a_1, a_2, a_3]^\mathsf{T} = \Theta \cdot \boldsymbol{\omega}$, where $\boldsymbol{\omega}$ is a unit vector representing rotation axis, Θ rotation angle and it holds $\|\mathbf{a}\|_2 = |\Theta|$. Every rotation matrix can be expressed in terms of angle-axis representation as an exponential map:

$$\mathbf{R} = \exp \mathbf{a} = \mathbf{I} + \sin(\Theta)\,[\boldsymbol{\omega}]_\times + (1 - \cos(\Theta))\,([\boldsymbol{\omega}]_\times)^2 \tag{2}$$

Conversely one can obtain angle-axis from the rotation matrix representation, through a logarithm map:

$$\mathbf{a} = \log \mathbf{R} = \Theta \cdot \boldsymbol{\omega}, \tag{3}$$

where $\Theta = \arccos(\frac{\mathrm{trace}(\mathbf{R})-1}{2})$, $\boldsymbol{\omega} = \frac{\tilde{\boldsymbol{\omega}}}{\|\tilde{\boldsymbol{\omega}}\|_2}$, $\tilde{\boldsymbol{\omega}} = [\tilde{\mathbf{K}}_{3,2}, \tilde{\mathbf{K}}_{1,3}, \tilde{\mathbf{K}}_{2,1}]^\mathsf{T}$ and $\tilde{\mathbf{K}} = \frac{\mathbf{R}-\mathbf{R}^\mathsf{T}}{2}$.

Let's return to constructing new geometry-aware distance metric and denote $\mathbf{R} \in \mathbb{R}^{3\times3}$ and $\hat{\mathbf{R}} \in \mathbb{R}^{3\times3}$ two rotation matrices between

Here $[\boldsymbol{\omega}]_\times$ denotes the cross product matrix of $\boldsymbol{\omega}$, i.e. $\boldsymbol{\omega} \times \mathbf{v} = [\boldsymbol{\omega}]_\times \mathbf{v}$ for every $\mathbf{v} \in \mathbb{R}^3$.

which we measure distance. The standard Euclidean distance between them is as follows:

$$d_M(\hat{\mathbf{R}}, \mathbf{R}) = \left\|\hat{\mathbf{R}} - \mathbf{R}\right\|_F . \tag{4}$$

Let's recall, that a product of two rotation matrices, $\hat{\mathbf{R}}$ and $\mathbf{R}$, is the rotation of the difference angle between them. For the sake of simplicity, let's define $\mathbf{u} = [u_1, u_2, u_3]^T$ through $\mathbf{U}$ as:

$$\mathbf{U} = \frac{\hat{\mathbf{R}}\mathbf{R}^T - \mathbf{R}\hat{\mathbf{R}}^T}{2} = \begin{bmatrix} 0 & -u_3 & u_2 \\ u_3 & 0 & -u_1 \\ -u_2 & u_1 & 0 \end{bmatrix}.$$

Logarithm map can also be expressed in the following form, where the l_2-norm of it is the geodesic loss.

$$\log \hat{\mathbf{R}}\mathbf{R}^{\mathbf{T}} = \mathbf{a} = \boldsymbol{\omega} \cdot \Theta = \mathbf{u} \cdot \frac{\arcsin(\|\mathbf{u}\|_2)}{\|\mathbf{u}\|_2} \tag{5}$$

$$d_G(\hat{\mathbf{R}}, \mathbf{R}) = \left\|\log(\hat{\mathbf{R}}\mathbf{R}^{\mathbf{T}})\right\|_2 = |\arcsin(\|\mathbf{u}\|_2)| = \left|\arcsin(\frac{\|\mathbf{U}\|_F}{\sqrt{2}})\right| \tag{6}$$

2.8 WEIGHT SHARING

Implemented model has the ability to use shared weights for input layers, i.e. $\mathbf{W}^e = \mathbf{W}^d = \mathbf{W}^f = \mathbf{W}^c$ as well as using shared RNN cell between encoder and decoder $\text{RNN}^e = \text{RNN}^d$ (latter has been shown in [4] to help with speeding up the convergence) . In general we found that weight-sharing does not have a big impact on overall performance, however it helps significantly with improving training stability (which can be contributed to smaller number of trainable parameters).

3 RESULTS

Our best performing model consists of sequence-to-sequence architecture. A single LSTM cell of size 1024 is used (in most of our experiments we observed that LSTM outperfomed GRU, which is contrary to findings from most of the papers we used as a reference). Before passing inputs to RNN we embed them to hidden state dimension of size 1024. We make use of residuals connections and geodesic loss. Data (in rotation matrix representation) was standardized. RNN and input weights in encoder and decoder were shared. During training we used batch size 100, learning rate 0.0005 with exponential decay, gradients were clipped to a maximum $l2$-norm of 5.0 and ran the model for 150 epochs. Sampling loss was not deployed (empirically we found it to have a negative impact on performance, at least as far as minimizing the angular metric is concerned). We also tested using bi-directional encoder, however we did not observe improvement in performance.

Table 1: Evaluation

Model	Public Score
Zero-velocity baseline	3.6162
AGED 1LR-LSTM 1024(s2s) 256(disc) GEO	3.5418
SEQ2SEQ 4LR-GRU RES STAND GEO	3.3002
SEQ2SEQ 1LR-LSTM RES STAND GEO	**3.2692**

3.1 COMMENTS ON TRAINING

While running different experiments we ran into multiple issues. In the following we outline our approach to tackling those.

- *Training instability:* Mostly we were using two optimizers, SGD and Adam. With latter we observed quick convergence, however as the training loss got closer to zero, huge spikes started to appear in the training loss curve, making the training process less stable. Spikes appeared periodically, which could be seen as a sign of certain batches including tougher input examples. To get around it, we increased batch size and also played around with ϵ parameter of Adam optimizer (parameter which is supposed to help with numerical instability for smaller values of gradients). With SGD, loss curves were much smoother, however convergence was slower and usually it was not able to obtain same level of performance as Adam.
- *Steep decrease in gradients:* We often observed both, training and validation loss, plateauing early on in the training (after only a couple of epochs). To investigate this we visualized max global norm of gradients at each global step and found out that it decreases very steeply from values around 100 to values within 0.2-0.4 range, where it stays for the rest of the training. To make learning more "spread out" we tried different weight initializations (Xavier, He).
- *GANs training:* When running version of AGED model[3], we noticed that validation loss suddenly explodes. After visualizing losses of generator (in our case sequence-to-sequence predictor) and both discriminators separately we realized that is due to discriminators becoming too strong (their training loss converged to zero, i.e. they were able to distinguish between predictions and ground-truths perfectly). To alleviate this, we played around with decreasing discriminators' cell size and with updating their weights only every second or third global step, and also by increasing number of layers used in predictor (in order to make it more powerful).
- *Overfitting:* For some experiments we observed that training loss quickly converged to 0, whereas validation loss got stuck. To combat this we deployed standard techniques such as dropout (both in linear layers and in RNN cells) and $l2$ weight initialization.

3.2 CODE

Our code, together with instructions for replicating results, is available at https://gitlab.inf.ethz.ch/COURSE-MP19/Slovenia. It has the capacity to replicate wide range of different seq2seq models along with training hyperparameters. It includes components with which it is possible to reproduce two papers with (or close to) state-of-the-art results ([3], [4]). We note that results we obtained using those architectures are worse than the ones presented in the papers. We would contribute this to either a lack of deep learning fine-tuning skills on our side or a semantic bug hiding somewhere in our code skeleton. Huge fine-tunning and debugging efforts in order to resolve either of the two aforementioned problems, unfortunately did not yield the desired effect of our model achieving the expected result.

REFERENCES

[1] Katerina Fragkiadaki, Sergey Levine, Panna Felsen, and Jitendra Malik. 2015. Recurrent Network Models for Human Dynamics. arXiv:arXiv:1508.00271

[2] Partha Ghosh, Jie Song, Emre Aksan, and Otmar Hilliges. 2017. Learning Human Motion Models for Long-term Predictions. arXiv:arXiv:1704.02827

[3] Liang-Yan Gui, Yu-Xiong Wang, Xiaodan Liang, and Jose M. F. Moura. 2018. Adversarial Geometry-Aware Human Motion Prediction. In *The European Conference on Computer Vision (ECCV)*.

[4] Julieta Martinez, Michael J. Black, and Javier Romero. 2017. On human motion prediction using recurrent neural networks. arXiv:arXiv:1705.02445