RWTH AACHEN UNIVERSITY

The present work was submitted to the
Chair of Computer Science 13 (Computer Vision)
Faculty of Mathematics, Computer Science and Natural Sciences
Prof. Dr. Bastian Leibe

Master Thesis

Temporal Modeling of 3D Human Poses in Multi-Person Interaction Scenarios

presented by

Stefan Erlbeck
Student ID: 344628
2021-12-16

First examiner: Prof. Dr. Bastian Leibe
Second examiner: Prof. Dr. Leif Kobbelt

Abstract

Anticipating human movement is vital for autonomous systems in order to ensure safety and understand human intentions. Although a variety of methods to solve motion prediction with deep learning have been proposed, we argue that existing research lacks focus on two key aspects. Many methods remove the global components of motion and treat each human individually, which oversimplifies the dynamics of human motion. In order to demonstrate that claim, we adopt a successful existing method using recurrent neural networks to additionally model global motion. This model serves as a baseline to our new method based on attention, which is used both for reasoning about time and interactions. For our evaluation, we enhance the quality of existing multi-person datasets by fusing different views. Our extensive evaluation shows that pose representation and normalization is crucial to successful models and we highlight the difficulty of modeling global positioning in combination with relative motion. Nevertheless, the adopted recurrent method comes close to state of the art performance which was trained without global motion. We furthermore solidify our claim that frameworks should be aware of other humans in the scene by showing that our attention-based method is able to close the performance gap to the recurrent method when considering multiple subjects jointly.

Contents

1 Introduction

Humans naturally deduce intentions from motion cues during all kinds of interactions, for example greeting usually involves shaking hands, where failing to understand these intentions might lead to situations perceived as awkward if one person refrained from accepting the hand shake, e.g., due to safety concerns during a pandemic. This process of motion forecasting can be formalized as the task of extrapolating an observed human motion into the future. The ability to predict human goals based on observed motion is crucial when participating in traffic since cyclists often indicate their intention to turn by raising a hand. With recent breakthroughs in computer vision and natural language processing comes an increasing interest in admitting autonomous cars and even humanoid robots outside of controlled environments. This process can only be approved if the safety of all involved humans can be ensured. In order to achieve this goal, it is key to automatize the prediction of human motion as an important ingredient to inferring human intentions.

1.1. Applications

A wide variety of activities requires anticipation of movement. For humans, motion prediction is probably most relevant in sport activities like handball where correct forecasting is vital to winning. Automated prediction is still far off to model complex feints but might one day be used to analyze opponent behavior prior to a match. However, current methods could be applied in environments where feinting is detrimental. For example, pedestrians do not hide their goals in order to avoid collisions, so that autonomous vehicles need to infer these goals. Similarly, autonomous agents need an overview of other traffic participants through tracking. However, pedestrians may temporarily be occluded behind objects in which case motion forecasting can be used to bridge these gaps. Likewise, understanding intentions is crucial to navigating through crowds. Moreover, real-time performance is essential, but complex frameworks suffer from a small reac-

tion lag due to the heavy computational load. Motion prediction can counteract this lag so that the world model is less outdated.
One goal of robotics is to create robots which are able to interact with humans, e.g., advising customers or freeing nurses of straining tasks like supporting mobility-impaired patients. In these scenarios, the robots need to be able to understand human motion to ensure safety. Lastly, applications in virtual reality, augmented reality and entertainment should be mentioned.

1.2. Motivation

Although the field of motion prediction is relatively new, a lot of different methods have already been proposed to tackle the problem. Many authors seem to be interested in generating realistic motions for time horizons of more than 10 s. While the problem is challenging, we argue that these long time horizons are not useful for autonomous systems because of the inherent ambiguity, i.e., solutions are just guesswork for such a time horizon. In contrast, modeling multiple humans jointly has only recently caught the attention of the research community. Although single-person motion forecasting may suffice for direct interactions of a robot with a human, the robot also needs the capability to navigate in scenarios with multiple people. In such cases, important constraints on the positioning of each individual can be derived from understanding interactions correctly, e.g., that palms touch during a handshake. These constraints unlock their full potential when all humans are modeled within the same fixed coordinate system. However, many existing approaches model motion in a coordinate system centered to the human, which completely removes global orientation and positioning with respect to the camera. A truly autonomous system would therefore require a second framework for inference of global 3D positioning. This seems far from optimal to us: if two frameworks have to be used for highly correlated tasks, these systems should work together instead of separately. But without global motion modeling, there is no possibility for refining a 3D pose detection hypothesis and vice versa. We therefore argue that motion modeling should be extended to a global setting. Furthermore, we hypothesize that global positioning is refined from multi-person modeling due to the aforementioned constraints from interacting humans.

1.3. Problem Definition

Let us first formally define the problem of motion prediction. Assume that the human skeleton consists of J body joints, where each body joint is a point $x_j \in \mathbb{R}^3$ in three-dimensional space. We can then represent the complete human pose at a fixed point in time f as $p_f = (x_1^T \ \dots \ x_J^T)^T \in \mathbb{R}^{J\times 3}$. Then a motion is a sequence of poses $(p_1, \dots, p_F)$ for equidistant points in time $1, \dots, F$. We also

denote these points in time as frames, which is why they are subscripted with the letter f. The task of motion prediction then becomes the problem of inferring $(p_{F+1}, \ldots, p_{F+H})$ from $(p_1, \ldots, p_F)$ for a fixed time horizon H. We usually choose F and H such that the method is exposed to 2 s of known historic poses, and needs to extrapolate 1 s into the future. These are prominent values in the literature and more importantly, constitute a time horizon where the inherent ambiguity of the task is still limited, i.e., we can assume that the majority of probability mass is distributed across a set of almost identical future pose sequences.
We will also refer to future poses as target poses in analogy to regression targets and refer to input poses as conditioning poses.

1.4. Outline

In order to understand our approach to motion prediction, a solid background on recent techniques and architectures in deep learning is required, especially concerning recurrent neural networks and attention. We will therefore provide an overview of basics techniques in deep learning in Chapter 2, along with references for further information. Furthermore, we will look at more details on human pose representations and shortly explain the most relevant mathematical concepts needed for that.
After the basics are laid out, we will present related research in Chapter 3. This chapter is split into three parts. First, we will pin down the role of motion forecasting inside computer vision by introducing related tasks like 3D pose detection and highlight similarities and differences. Next, we will look at motion forecasting from a syntactical point of view and explain how sequences are modeled and generated. Lastly, we will present an overview of existing methods for motion forecasting.
Our own approaches are explained in Chapter 4 and Chapter 5. In Chapter 4, we adopt an existing system using recurrent neural networks. That approach is widely considered to be a milestone in the field of motion forecasting. There are two reasons behind this step. Firstly, we want to demonstrate how to extend the task to additionally capture global positioning and orientation. Secondly, we want to create a strong baseline method which is necessary because existing approaches to multi-person motion prediction are often hard to compare to due to uncommon metrics or datasets. Our own method using the attention mechanism is presented in Chapter 5. We will demonstrate how attention replaces the temporal dependency of recurrent approaches and also how it generalizes to multi-person modeling with little additional effort. Due to these advantages, we argue that attention-based architectures are preferable once they obtain the same accuracy as recurrent ones.
Multi-person datasets with accurate 3D ground truth poses are currently scarce. Although there are several efforts to alleviate this gap concurrently to this work, they were not published in time. We therefore dedicate Chapter 6 to our approach

of obtaining refined ground truth data from datasets recorded with commodity hardware. This is of course far from optimal and should therefore be considered as a placeholder for upcoming datasets.
Consequently, we split our evaluation described in Chapter 7 into an analysis on high-quality data and on our refined data. Chapter 8 shows the result of our extensive tuning on the AMASS dataset, which contains single-person motions from professional motion capturing systems. Furthermore, we will compare our two methods to several heuristics and a state of the art method using attention as well. We will show that our approaches cannot fully compete with the state of the art, likely caused by the fact that the mentioned method was not required to model global motion as our methods were. However, we will also see that the gap to state of the art performance is rather small. Lastly, we will see that our attention-based method has trouble to keep up with our recurrent method for certain evaluation metrics and that different architectures exceed at global motion and relative motion. The second part of our evaluation is presented in Chapter 9, where we will move to the refined multi-person dataset. We will see that the performance advantage of our recurrent method is completely negated by the ability of attention to model multiple people jointly.
Lastly, we will discuss our results in Chapter 10 and explain how future research might improve our attention-based method.

Our contributions can be summarized as follows:

1. Extending an existing successful method to model global orientation and position.

2. Devising a new method that generalizes to multi-person motion forecasting with no constraints on the number of subjects.

3. Refining existing datasets in order to adequately evaluate our method.

4. Demonstrating the difficulties of global modeling and how multi-person approaches can alleviate these difficulties.

2 Fundamentals

In order to follow the next chapters, a strong background in deep learning is helpful. We will therefore dedicate this chapter to shortly explaining the most fundamental learning concepts relevant to our research. We will furthermore provide references for more in-depth explanations. Lastly, we also want to introduce common mathematical concepts and algorithms which are sometimes needed when processing motion data.

2.1. 3D Modeling

In order to model humans in three-dimensional space, some common mathematical concepts like distance functions and rotations are required. More importantly, one needs to understand how a human pose is actually represented, so that we will provide the details in this section.

2.1.1. Distance Functions

A metric or distance function is a function $d(x, y)$, which is positive definite, symmetric and fulfills the triangular inequality, in other words

$$\begin{aligned} &d(x,y) \geq 0 \quad \text{and} \quad d(x,y) = 0 \Leftrightarrow x = y \\ &d(x,y) = d(y,x) \\ &d(x,y) \leq d(x,z) + d(z,y) \end{aligned}$$

for all elements x, y, z from the metric space. Common metrics include the Manhattan distance and the Euclidean distance. The former is given by

$$d(x,y) = \|x - y\|_1 = \sum_{i=1}^{3} |x_i - y_i|$$

for two points $x, y \in \mathbb{R}^3$, whereas the squared Euclidean distance is given by

$$d(x,y)^2 = \|x-y\|_2^2 = \sum_{i=1}^{3}(x_i - y_i)^2$$

Squaring the Euclidean distance is common in machine learning, because often, derivatives need to be computed.

2.1.2. Rotations

Formally, a rotation is an orthonormal basis exchange of a coordinate system, so that applying a rotation results in the coordinate axis pointing into different directions compared to the original coordinate system. We will only consider rotations in three dimensions throughout this work. Together with translations, i.e., shifts of the origin and all other points along the same parallel vector, rotations form the group of Euclidean transformations which preserve distances, angles and areas [HZ03]. Rotations can be expressed in a variety of different representations. For the scope of this work, it suffices to view a rotation in three dimensions as a matrix $R \in \mathbb{R}^{3\times3}$ which satisfies $RR^T = R^TR = I$ where I is the identity matrix. The set of these matrices R form the so-called orthogonal group and those elements with determinant equal to one form a subgroup called special orthogonal group. This subgroup coincides with the rotation matrices. In contrast, those orthogonal matrices with determinant equal to minus also contain a reflection component, i.e., they change the coordinate system from right-handed to left-handed, which is usually not desired. For more information on rotations, and especially on alternative representations, we refer to [SSVO10].

2.1.3. Human Poses

In order to model human motion, one needs to be able to represent the human skeleton adopting different poses at discrete time steps. A sparse way to approximate human poses is to model the relationship, i.e., angles, between the most important joints like shoulders and elbows. This bears similarity to how robotic arms, etc., are modeled and is also adopted in animation software. In robotics, the most common way to accomplish that is to model each joint as a local coordinate system and the current pose configuration as a chain of rotations, which is sometimes referred to as joint angle representation. If one additionally knows the length of the limbs, one can then compute where an end joint is in the coordinate system of an ancestor joint by recursively chaining the rotations and translations of all joints in between. This process is called forward kinematics. Such an approach is also adopted by Loper et al. [LMR+15] in their effort to model human poses as well as body shapes in a single differentiable framework compatible with animation pipelines, see Section 3.1.1 for details on SMPL. An alternative approach to this is to model joints as points in the same coordinate

system, i.e., to model joint locations. The formulation as angles makes it easier to encode constraints like fixed limb lengths and impossible joint angles so it is advantageous for controlling robots. However, the purpose of human modeling is different to controlling actuators in a robot. Usually, the goal is to either visualize the poses or to integrate them into a 3D scene model. For both of these goals, joint angles are meaningless so that forward kinematics need to be applied anyways. We therefore argue that as long as the deep learning framework is able to reliably generate feasible poses, joint locations are more efficient as an underlying representation.
More background on kinematics can be found in [SSVO10].

2.2. Deep Learning

In the following, we will shortly describe important neural network architectures which are utilized in this work, and algorithms related to training them.

2.2.1. Multi-Layer Perceptrons

A broad class of neural networks is named multi-layer perceptrons (MLP). A single layer of such a network maps an input vector x linearly to $Wx + b$, where W is a learned weight and b a learned offset called bias. In this context, learning means optimizing under a certain loss function which defines what the output of the MLP should look like. Stacking multiple such layers would not make sense without the introduction of non-linear mapping, e.g., $\sigma(x) = \max\{0, x\}$, between them, since the composition of two linear functions is again linear. Two-layer perceptrons have been shown to be capable of approximating every function, if the dimension of the intermediate output is high enough [SSBD14]. In practice, more than two layers are commonly stacked instead of choosing a large hidden state.

2.2.2. Training

Neural networks are typically supervised using a loss function which measures how correct the outputs are. An example for a loss function often used in regression is the squared Euclidean distance between predicted vector and target vector. Of course, this assumes that training data exists for which the ground truth target is known. This assumption is never violated throughout the scope of this work.
A common iterative algorithm to optimize the weights of a neural network is gradient descent. The idea is that the gradient of the loss with respect to the weights points into the direction of steepest ascent, so that moving in the opposite direction produces a lower loss. The corresponding update rule is

$$w_{\text{new}} = w_{\text{old}} - \eta \frac{\partial L}{\partial w_{\text{old}}}$$

where L is the loss function with weight w_{old} and η is the learning rate governing convergence speed. Modern deep learning systems often apply optimized versions of gradient descent, e.g., Kingma et al. [KB15] average over past gradients for a less noisy estimate and additionally scale the gradients for more stable convergence.
It should be noted that the computation of the gradient is not trivial. The gradient vectors have to be computed in a smart order to efficiently reuse intermediate results. This is summarized in an algorithm called backpropagation. For more details, refer to [SSBD14].
Gradient descent originally computes the loss over all samples in the training dataset. However, this implies that the optimization might get easily stuck in a local optimum [Bis06]. Stochastic gradient descent only computes the loss over a single data point, which alleviates the previous problem but can be extremely slow for large datasets. As a consequence, most approaches use a middle ground where multiple data points selected at random are grouped into a (mini-)batch and each loss computation and weight update are performed on such a batch. Using this approach allows to implement optimization via efficient matrix products, and is less noisy than stochastic gradient descent. Mini-batches can be employed to modern alternatives for gradient descent as well.

2.2.3. Recurrent Neural Networks

Multi-layer perceptrons and most other feed-forward networks have a distinctive limitation. Each input token produces a deterministic output, irrespective of previous inputs. This is problematic for applications like time-series prediction where the input consists of multiple measurements at different points in time. Recurrent neural networks (RNNs) add a temporal connection between each layer. The naive implementation computes the output as $h_t = \sigma(W_h h_{t-1} + W_x x_t + b)$, i.e., the output depends on the current input x_t and on the previous output h_{t-1}. Such an RNN can be notoriously hard to train because gradients might grow or shrink uncontrollably in magnitude, compare Hochreiter et al. [HS97]. Therefore, they devised an RNN design with improved gradient flow. The key idea of their method, which is called long short-term memory (LSTM), is to use gates of the form $\sigma(W[h_{t-1}, x_t]+b)$ where square brackets denote concatenation. The result is a vector containing values only between zero and one due to the sigmoid function σ. This vector is then used to control the access to the internal state of the LSTM cell, e.g., which parts of the cell state should be forgotten (set to zero). An alternative to LSTM with a simplified design is named gated recurrent unit (GRU) and was presented in [CvMG+14].

2.2.4. Attention Mechanism

While RNNs governed state of the art in many sequence tasks like natural language processing for years, they suffer from low parallelization potential. Furthermore, the limited size of the cell state poses an inherent bottleneck to keeping both short-term and long-term dependencies. Vaswani et al. [VSP+17] proposed a different approach to sequential modeling called attention. It is computed as

$$\text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

where the rows of matrices Q, K and V are query, key and value vectors and the denominator contains a scalar rescaling factor. Essentially, the formula computes a weighted sum of the value vectors for a query vector q_i where the weight of each individual value v_j is determined by the similarity between its query q_i and key k_j.
We can immediately observe two advantages of attention. Firstly, the information exchange between the value vectors is independent of the distance between i and j, which means that attention considers the relationship between distant inputs just as much as short-term dependencies. Secondly, the number of queries is independent of the number of key-value pairs. In fact, queries and key-value pairs do not even need to represent the same type of data, e.g., queries could represent words and key-value pairs could represent image features if one was to evaluate whether a sentence describes an image.
Vaswani et al. also made further contributions. Multi-head attention is an extension where attention is applied several times in parallel in order to explore more than one type of similarity between tokens. Since attention has no sense of token order, the absolute position of each token is encoded through sine and cosine waves with different periodicities. This positional encoding is added once to the input. Their resulting network trained on language translation is called transformer, although that term is sometimes also used for vastly different architectures containing an attention module.

2.2.5. Common Techniques

Neural networks are known for a considerably amount of time, but only recently became popular in a large variety of tasks. Additionally to higher computational resources, this may partly be accounted to a number of tricks which often seem to help in practical applications. Dropout multiplies random entries of the intermediate outputs with zero, which can be seen as a way to train multiple models and combining their benefits. Data augmentation diversifies the training dataset by applying modifications to the training data which the network should be able to deal with. Another trick is called weight decay, where the loss is extended to contain the norm of the network weights. This enforces small weights except where needed for a good performance. In order to see why small weights are

preferable, consider the case where a certain feature is only weakly correlated with the outcome. If the corresponding weight was large, fluctuations in that feature would be enhanced despite having little meaning to the outcome. Other techniques include residual connections, where the output of a layer (or multiple layers) is added to the input, so that it effectively learns to generate a correction of the input. These residual connections can greatly improve gradient flow which allows for deeper models. Lastly, consider the layer presented by Ba et al. [BKH16], which re-normalizes a feature vector so that the different features are located around zero. This is for example helpful when using the rectified linear unit (ReLU) $\sigma(x) = \max\{0, x\}$ non-linearity. Imagine that all inputs were positive numbers. Then multiple linear layers would collapse into a single linear function and the convergence would be slow until the network learned to offset the bias to positive values.
Apart from the information about layer normalization, most of this section is based on [GBC16], which also serves as a reference for more details.

2.3. Algorithms for Preprocessing

Apart from deep learning, we require a few classical algorithms, which will be presented in this section. Most of these algorithms are used for preprocessing of noisy data.

2.3.1. Linear Assignment

The problem of linear assignment considers n workers which need to be assigned to m jobs. Each possible pair of worker and job induces a certain cost based on some cost matrix $C \in \mathbb{R}^{n \times m}$. The task is to find a non-extendable matching of jobs to workers, so that either no worker is unemployed or all jobs are being worked on. The task can also be formulated as finding a non-extendable matching with minimal cost. While many algorithms have been devised, some of the most efficient methods can be traced back to the Hungarian algorithm and use shortest augmenting paths to solve the problem [Cro16]. That means that in each step, a previously unassigned worker is matched to a job which was already assigned. Consequently, the original worker needs a new job, which can again be an assigned job. This is repeated until a previously assigned worker is matched to a previously unassigned job. Such an extension of an existing matching is called an alternating path and the algorithms look for the shortest of all alternating paths. Crouse [Cro16] derived such an algorithm which is also used in the scope of this thesis.

2.3.2. Sequence Alignment

The problem of sequence alignment consists of finding correspondences between the items of both sequences, where correspondences must never go backwards,

e.g., if a_i was matched to b_j, then no item after a_i can be matched to any item before b_j and vice versa. The problem can be solved efficiently with dynamic time warping (DTW) which utilizes dynamic programming to check whether a matching until (a_i, b_j) should be extended by matching b_{j+1} to a_i or to a_{i+1} or matching b_j to a_{i+1}. Salvador et al. [SC04] present a linear approximation of DTW called FastDTW, which recursively finds coarse alignments and refines those. Although Wu et al. [WK20] report that FastDTW is slower than full DTW, we observed considerably lower execution times, so that we will employ FastDTW.

2.3.3. Estimating Rotations

Consider two sets of vectors in $\mathbb{R}^3$ with the same cardinality. The task is to find a rotation matrix that, when multiplied with the vectors from the first set, produces the vectors in the second set. Obviously, this problem has no solution in almost all cases, so that one instead tries to minimize the squared Euclidean distance between the rotated vectors and the vectors in the second set. The problem was solved by Kabsch [Kab78] and essentially comes down to stacking the vectors into matrices A and B, computing $B^T A \in R^{3\times3}$, applying singular value decomposition $B^T A = U\Sigma V^T$ (with sorted singular values) and choosing UV^T as the solution to the problem. The algorithm can be applied to point clouds by shifting the center of mass to the origin first.
However, the rotation matrix UV^T may have a determinant of minus one. This means that the vectors were actually reflected and rotated and is often not a feasible solution. Kabsch observed that multiplying the last column of V by minus one in that case alleviates the problem.
Another difficulty is that the data may contain a large number of outliers, i.e., corrupt data points which should be ignored. In order to increase robustness, one can use the meta-algorithm random sample consensus (RANSAC) [FB81]. Instead of computing the hypothesis (in this case the rotation matrix) from all data points, one randomly samples a minimal number of data points such that the solution is unique, which is three vector pairs in our case. Next, one computes the hypothesis on this subset and evaluates how many points in the whole dataset support it, i.e., how many vectors lie within a reasonable error threshold. These steps (sampling and evaluation) are then repeated for multiple iterations. In the end, the best hypothesis, i.e., the one with the highest support, is used to divide the dataset into inliers and outliers so that one can then choose the hypothesis minimizing the error on all inliers.

Although implementations of Kabsch' algorithm exist in common libraries, we ran into the problem of having to compute a large amount of such rotations. We therefore implemented a batched version of Kabsch' algorithm based on the singular value decomposition from PyTorch.

2.3.4. Geometric Median

For a point cloud $x_1, \ldots, x_m \in \mathbb{R}^n$, the problem of finding the geometric median c is defined as

$$\operatorname*{argmin}_{c \in \mathbb{R}^n} \sum_{i=1}^{m} \|x_i - c\|_2$$

which is a generalization of the one-dimensional median. If the Euclidean norm was squared, one could simply compute the center of mass, i.e., the arithmetic mean component-wise. The proof is analogue to the maximum likelihood estimation of the mean of a multivariate Gaussian, compare for example [Bis06]. Without squared norms, the problem is strictly convex, i.e., every local minimum is also global [VZ00]. Nevertheless, iterative algorithms like [VZ00] are required. The basic idea is to compute the center of mass, but value points higher when they were farther away from the previous estimate. However, finding a good initialization is not trivial.

3 Related Work

Motion Forecasting is one of many computer vision problems involving the modeling of human behavior. We will therefore present a brief overview of tasks emerging around human modeling, where we will highlight similarities and differences to motion prediction. Next to modeling human behavior, the other key aspect of motion prediction is the generation of a sequence. Therefore, we will describe common approaches to this class of problems and analyze benefits of drawbacks of these approaches.
After providing these overviews, we can then proceed to look at state of the art methods solving motion prediction in order to understand which key developments have been made.

3.1. Related Tasks

Motion prediction shares similarities and methodologies with many other tasks from computer vision and even natural language processing. In this section, we will provide a brief overview of related problems in deep learning. Problems might share semantic and syntactic similarities, which we will both consider in Section 3.1.1 and Section 3.1.2 respectively.

3.1.1. Modeling Humans

Semantically similar tasks to motion prediction mainly occur in computer vision. Motion forecasting is a task where reasoning about human behavior is key. Many related problems also require knowledge about human appearance, e.g., skeletal representations are often extended to human shapes via frameworks like SMPL [LMR+15], more details later. We introduce a taxonomy of semantically related tasks in Figure 3.1 based on the structure of the problem solution.

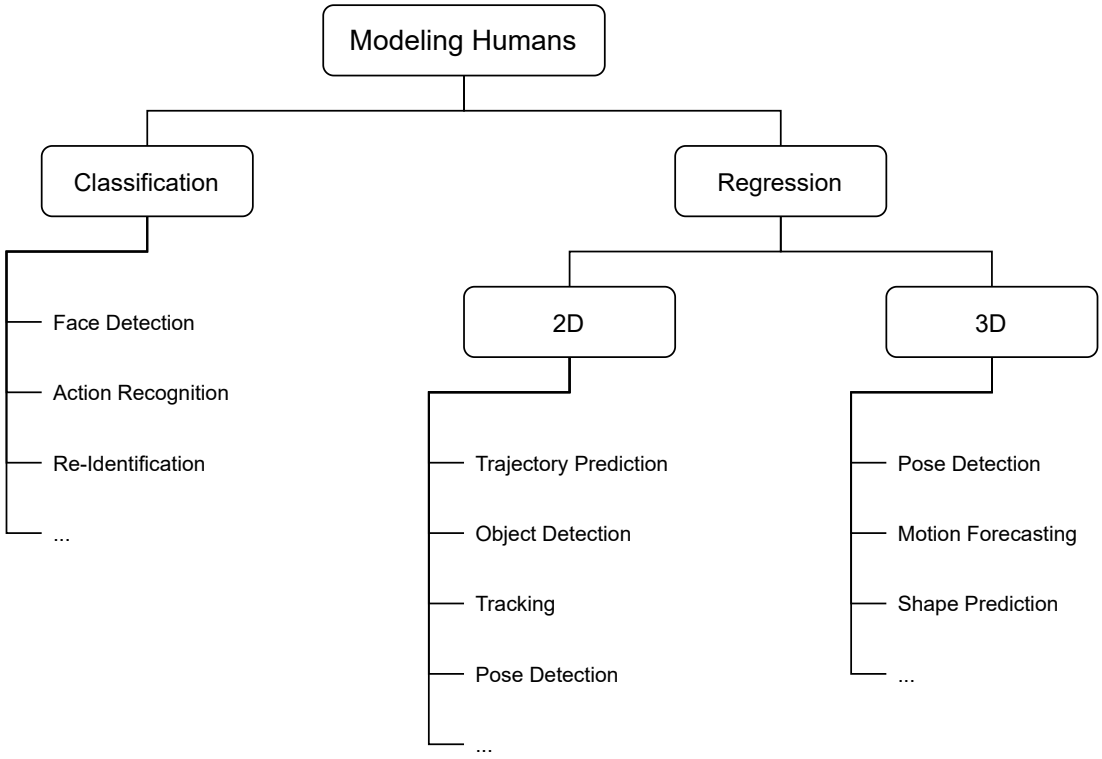


Figure 3.1.: Taxonomy of computer vision tasks related to humans.

Let us first consider classification problems involving human appearance. An early example for this problem class is face detection, i.e., the binary decision of whether an image shows a face or not. Viola et al. [VJ01] proposed a famous and efficient method for face detection using a hierarchy of simple decision rules based on image features. This work dates back to when deep learning was not yet popular.

A more recent and more complex example of human appearance classification is the task of re-identification, where a queried human needs to be matched against a database of individuals. Wojke et al. [WB18] derived a type of dense layer that forces the network backbone to learn feature representations which can be compared with cosine similarity. This is beneficial because it allows comparisons with unknown humans. We will later use their approach during our data preprocessing.

Like human appearances, human behavior can also be classified. One example for that is action recognition. Here, an RGB video depicting human behavior is shown to the classifier which then needs to assign the video to a class of human actions, e.g., eating, walking, etc.. One can think of action recognition as the classification variant of motion forecasting, because both problems consist of understanding human motions. A popular method for action classification

was presented by Carreira et al. [CZ17], who demonstrate that pre-training on large action recognition datasets benefits the accuracy greatly. They furthermore present an approach to benefit from image classification networks which are considerably more advanced than those prevalent in action recognition.

On the other hand, regression tasks usually require multiple continuous outputs. These tasks can further be divided into 2D, i.e., the regression method predicts pixel location, and 3D, where the goal is to predict locations in world space. We will first consider 2D regression problems and then their 3D variants if existent. A low-dimensional 2D regression task is trajectory prediction, where a past trajectory is known and needs to be interpolated into the future. Humans are simplified to moving points on a 2D plane and the difficulty comes from modeling social interactions like people moving in groups. We note that this is quite similar to global motion estimation, because both tasks involve the extrapolation of positions into the future. Trajectory prediction was solved for example by Alahi et al. [AGR+16], who utilize a single auto-regressive recurrent network to predict each trajectory individually. The key to their method is the intermediate social-pooling layer, where the hidden states of the RNNs are aggregated if the respective humans are close to each other.

Two highly common regression problems in the space pixel coordinates are object detection and tracking. Both involve the prediction of bounding boxes in order to mark objects in the image, e.g., by regressing the pixel coordinates of the box corners. These tasks are not necessarily related to humans, but important applications like autonomous cars utilize these techniques for pedestrians and other traffic participants. The difference between object detection and tracking is that tracking also consists of a temporal component because in each frame, the tracked objects need to be associated to those objects from the previous frame. A recent object detection approach using transformers was presented by Carion et al. [CMS+20]. Here, the image features are fed into the transformer encoder while the decoder maps "placeholders", which are called object queries, to bounding boxes. Attention is used as the connection mechanism between the object queries and the image features. Meinhardt et al. [MKLTF21] extend that transformer structure to perform tracking. This can simply be achieved by using auto-regressive feedback on the positive detections, i.e., the input to the decoder consists of empty object queries and filled track embeddings from the previous frame. We emphasize that tracking shares many similarities with motion forecasting. Both tasks are regression problems, include a temporal component and largely deal with human movement. However, the human appearance is much more important for tracking than for motion prediction. Moreover, motion forecasting has to interpolate longer into the future.

The regression task with the highest similarity to motion forecasting is pose

prediction, at least concerning how outputs look like. Pose prediction is the task of regressing the location of joints like wrists and feet. These locations can be regressed both in pixel (2D) or in world coordinates (3D). Again, we will consider the 2D variant first. One approach to multi-person pose estimation is presented by Cao et al. [CSWS17]. Two CNNs are iteratively applied to the image features. The first CNN predicts confidence scores for the different joints as heatmaps. However, association between joints becomes extremely hard for crowded scenes. Consequently, the second CNN predicts 2D vector fields for each point lying on a limb. For association, one can use the vector field to learn where the kinematic child joint lies. Since this method infers poses from joints, it is called bottom-up. Another line of work approaches the problem top-down, i.e., first detecting image crops containing a single human and then inferring joints. An example for such a top-down pose detection in 3D coordinates was presented by Sárándi et al. [SLAL21]. The authors directly generate volumetric heatmaps via convolutions from the input image. Next, the expected voxel location of each joint is computed. The key advantage is their translation-invariant joint loss which allows the network to place the root joint anywhere inside the heatmap. As a consequence of the decoupling of heatmap and image coordinates, a full pose can be regressed even for cropped image. If additionally the camera intrinsics are known, their method can be extended to compute the absolute position on top of the root-relative pose.
Although both motion forecasting and 3D pose detection both consist of the regression of human poses, there are also many differences. These include that for motion prediction, the input is not an image but a pose sequence and that not only the current pose should be regressed but also those after that. Current approaches to motion forecasting are discussed extensively in Section 3.2.

The last of the presented 3D regression problem involving humans is shape prediction, where the goal is usually to predict a sparse but high-dimensional surface mesh of the human body. As surface meshes are difficult to manipulate, shape prediction often involves the regression of the skeleton pose as well so that the surface mesh can then be modified, e.g., with animation software. A framework to accomplish that was presented by Loper et al. [LMR$^+$15]. Their model, called SMPL, contains learned deformations of the default surface mesh based on the body shape, e.g., how athletic a person is, and based on the human pose, in order to model soft tissue deformation around joints. These two factors, the body shape of a certain human, and its pose at a point in time can be controlled by two low-dimensional input vectors, which can for example be generated by a deep learning method. The key is that SMPL is able to apply forward kinematics according to the pose vector provided so that the surface mesh will adopt the corresponding pose.

After presenting this overview on related tasks, we want to mention that other

taxonomies are possible, e.g., based on the input data. However, we argue that the goal of inference is a more discriminative property than for example whether the image is colored. This taxonomy is also by no means complete, e.g., recent research often aims at segmentation or object detection from point clouds. While motion forecasting with joint locations comprises a point cloud as well, there are many distinctions. Firstly, motion forecasting is extremely low dimensional compared to a LiDAR scan. Secondly, our data is assumed to be free of occlusion and background, i.e., each joint location is always known and each point in the input can be mapped to a relevant joint. Finally, we aim to manipulate points, i.e., perform realistic transformations of human poses.

3.1.2. Sequence Generation

Syntactically, motion forecasting is a sequence-to-sequence tasks similar to language translation or trajectory prediction, where both input and output are sequences. We therefore now provide an introduction on how to approach this class of problems.
In general, input and output sequences cannot be assumed to have the same length. Thus, many authors, e.g., [VSP+17, AGR+16, LYRK21] among others, utilize encoder-decoder-architectures in order to decouple the input and output sequence. The encoder computes a latent feature representation of the input, often with fixed length, and the decoder performs sequence generation while being conditioned on the encoder output. For example, Vaswani et al. [VSP+17] apply these architectures to automated language translation where the encoder computes feature vectors for each word in the source language sentence and the decoder generates a translation conditioned on these words. Naturally, encoder and decoder do not share weights because, e.g., grammar can differ a lot between source and target language. Interestingly enough, the feature representation is nevertheless shared between the two languages by grouping common character combinations in both languages into new symbols, i.e., the input and output tokens are in fact not words but character groups. For brevity, we still refer to them as words.

While the encoder-decoder formulation solves the problem of decoupling input and output, it does not solve the problem of how to actually generate the output sequence. Therefore, we will now investigate methods to achieve that.
A popular method to generate a sequence of output tokens is to use an autoregressive connection. This means that the network is trained to transform a vector describing the state at time t_{i-1} to the vector describing the state at time t_i. Since the output comes from the same domain as the input, the network can then use the output, i.e., the estimated state at time t_i, to generate the estimated state at t_{i+1}. This procedure can then be repeated as often as necessary in order to generate longer sequences.

Auto-regressive methods can be subdivided into two categories based on how exactly training is performed. In teacher forcing, which is for example used in [AGR+16], the input sequence is the ground truth sequence shifted by one step. Consequently, to predict n steps into the future, the input is the ground truth at $t_0, t_1, \cdots, t_{n-1}$ and the target variable is the ground truth at $t_1, t_2, \cdots, t_n$. In sampling-based training, the only ground truth used is the feature vector at t_0. The estimated corresponding output $\hat{y}_1$ then becomes the input at t_1 and so one. Thus, the network is trained to work with its own output. Note that even with teacher forcing, inference has to be sampling-based as the future is obviously unknown.
Training with teacher forcing reduces the training duration because all inputs are directly known, which allows more aggressive optimization and parallelization even for recurrent models. Furthermore, it can lead to more stable training because the correspondence between input and output is always exact unlike in sampling-based training. Here, a complete mismatch between target y_i and estimation $\hat{y}_i$ leads to the supervision that $\hat{y}_i$ should be matched to y_{i+1}. But if $\hat{y}_i$ is a valid state different from y_i, one usually cannot assume that it should lead to state y_{i+1} as well. In contrast, sampling-based training can lead to more robust models because the network gets used to its own noise. Which of these two properties is more important depends on the task. Motion forecasting usually employs sampling-based training as Martinez et al. [MBR17] reported it to outperform teacher forcing. Prior to that, Fragkiadaki et al. [FLFM15] utilized teacher forcing with increasing amount of corrupting noise on the input, which Martinez et al. criticize as difficult to tune.

In their machine translation framework, Vaswani et al. [VSP+17] also use auto-regressive generation with teacher forcing. A key difference is that they additionally apply beam search, where instead of greedily generating the most likely word, several words with high probabilities are stored. This is useful because sometimes choosing a suboptimal word can lead to more likely sentences in the long run. In contrast, the amount of valid human poses is infinite so that the probability distribution of human poses cannot be represented as a probability vector. It therefore becomes rather sophisticated to model multiple futures similar to how multiple translations are explored, e.g., Ghosh et al. [GSAH17] report that the Gaussian mixture models they used lacked expressive power.

A completely different approach to sequence generation is presented by Hernandez et al. [HGM19]. The authors formulated motion forecasting as an inpainting problem. Inpainting is the task of generating image coloring for regions where the original data is lost, e.g., in order to remove text written on an image. If the task is to predict a time series, the task can equivalently be formulated as inpainting the future of a temporal tensor. Consequently, this approach is more

efficient both in training and inference because it does not depend on sequential computation like auto-regressive generation. But like all sequence generation approaches, it inherently suffers from ambiguity and error propagation the longer the target length is.

3.2. Motion Forecasting

We have now given an overview of tasks involving modeling of human appearance and behavior. Furthermore, we have presented different approaches to generate future motions, therefore it is now time to actually focus on motion forecasting. We will therefore present a wide variety of recent method specifically for that task.

An early milestone was presented by Martinez et al. [MBR17], who showed that previous attempts were often outperformed by a simple baseline heuristic called zero motion, where the last known pose is used as predictions. They devised a GRU architecture with comparably few weights. One key contribution is to employ a residual connection from input to output so that the network effectively models velocity. Ghosh et al. [GSAH17] presented a deeper LSTM. After each prediction, the output is fed to a denoising auto-encoder which was trained separately on denoising as well as joint reconstruction. This allows the model to produce realistic poses over a time horizon of 10 s, at least for periodic actions like walking. Liu et al. [LWJ+19] model the space of all human poses as a Lie group. The strong ties to the theory of Lie groups allow encoding of pose constraints like restrictions on bone lengths and the degrees of freedom in some joints. Their decoder is the usual auto-regressive RNN, but updated with the average of all hidden states from the encoder in order to not over-emphasize the last known frames. In contrast, Martinez et al. use the large hidden state of the encoder because they argue that the first prediction often lacks smoothness. Aksan et al. [AKH19] designed a custom hierarchical layer. For each joint, there is a dedicated small sub-layer conditioned on its kinematic parent and the global feature vector. This feature vector can be the hidden state of an arbitrary RNN, e.g., the GRU from [MBR17]. Du et al. [DVJR19] devised an LSTM for motion prediction with shapes instead of poses using the aforementioned SMPL framework [LMR+15]. They furthermore introduced additional loss terms derived from domain knowledge, e.g., symmetry of shoulders and ground plane constraints. Adeli et al. [AAR+20] model multiple humans at the same time. Their encoding LSTM is applied on each person separately. Afterwards, they employ social pooling as in [AGR+16]. Another key difference to previous methods is that they model motions in a fixed global coordinate system instead of a relative system following the subjects. Kundu et al. [KBM+20] designed a network specifically for long-term synthesis of partner dance motions. One part of the network is provided with the known poses of one person and predicts their dance partner from that. The other part of the network takes one person and predicts its future.

By alternating between these two steps, future motions of arbitrary length can be synthesized without losing the correlation between the two dance partners. They evaluate time horizons of 20 s.

Before the emergence of transformers [VSP+17], few methods for motion prediction used approaches other than RNNs. Notable exceptions are Hernandez et al. [HGM19] and Mao et al. [MLSL19]. Hernandez et al. formulate the problem as an inpainting task (compare Section 3.1.2) and employ two convolutional neural networks following the generative adversarial network approach. The task of the generator is to predict the future motion and the discriminator evaluates whether the result looks similar to real data. Their motivation for the network structure is that image inpainting is solved in the same way. Mao et al. choose a different approach. Given a motion sequence, they replicate the last known frames as often as it shall be predicted, like the zero motion baseline from [MBR17]. Afterwards, they apply the discrete cosine transform (DCT) on each joint independently, i.e., each joint trajectory is not represented as a temporal sequence of locations but as a weighted sum of cosine waves. The network then performs a refinement of these coefficients. Their motivation for using DCT is that those coefficients corresponding to high frequencies can be discarded after refinement to smooth the joint movement through space.

Recent approaches investigate attention mechanism to replace the recurrent connection. Cao et al. [CGM+20] first generate possible destinations and then predict the motion which the human undertakes to reach a destination, where the latter part is generated by the transformer. Like Adeli et al., their method is scene aware, i.e., the network is provided with image data in addition to the past poses. Li et al. [LYRK21] propose a transformer that generates dance movements conditioned on past poses and on music. They employ two transformers to embed audio features and pose features into the same latent space. These latent features are concatenated and used to generate future poses with a third transformer. Mao et al. [MLS20] also presented an extension to their DCT method. The network input does not only comprise the DCT coefficient, but also a prior estimate of future. This prior is obtained by motion attention, i.e., an attention mechanism compares how similar the last known poses are to older poses. If similar older poses are found, one can use their respective future (which still lies in the past) as a prior. This design is motivated by the fact that some motions like walking are periodic. Concurrently to our work, Aksan et al. [ACKH20] developed another transformer architecture. By now, it is peer-reviewed and published, but we still refer to a preprint. Their method divides a motion into a temporal and a kinematic dimension. Consequently, their transformer uses spatial attention to aggregate information between joints and temporal attention to achieve the same between different frames. The sequence is generated in an auto-regressive manner.

4 Forecasting with Recurrent Neural Networks

As our first method, we will adopt the approach to motion forecasting from Martinez et al. [MBR17]. Their method utilizes recurrent neural networks and is often used in research to obtain a reference performance. However, it ignores global motion and is therefore not directly comparable to our method presented in Chapter 5. To alleviate this, we adopt their architecture to global motion modeling. Furthermore, the adoption allows to analyze how much additional effort is required to transition from relative to global modeling.

4.1. Architecture

The encoder-decoder architecture from Martinez et al. [MBR17] is presented in Figure 4.1. The classification as an encoder-decoder model is rather conceptual because historic and future poses are fed to the same recurrent layer and consequently share the same internal state, though at different points in time. The key difference between the encoder and the decoder is that the encoder only updates the internal state while the decoder additionally passes the output of the last recurrent layer to the linear projection layer in order to generate a pose from the internal state. The output sequence is generated using the auto-regressive approach, which was extensively presented in Section 3.1.2.

The main differences to their model are threefold. Firstly, we represent poses as joint locations instead of joint angles, which we motivated in Section 2.1.3. Secondly, we stack multiple recurrent layers because we train on AMASS [MGT+19], which contains significantly more data (and therefore allows deeper networks) than Human3.6M [IPOS14] used by Martinez et al.. Moreover, AMASS contains a wide variety of challenging motions like somersaults, whereas Human3.6M contains 15 everyday actions of which a few are even redundant, e.g., walking, walking with a dog and walking as a couple. Lastly, Martinez et al. recommended to concatenate action labels to the network input as their action-agnostic network was

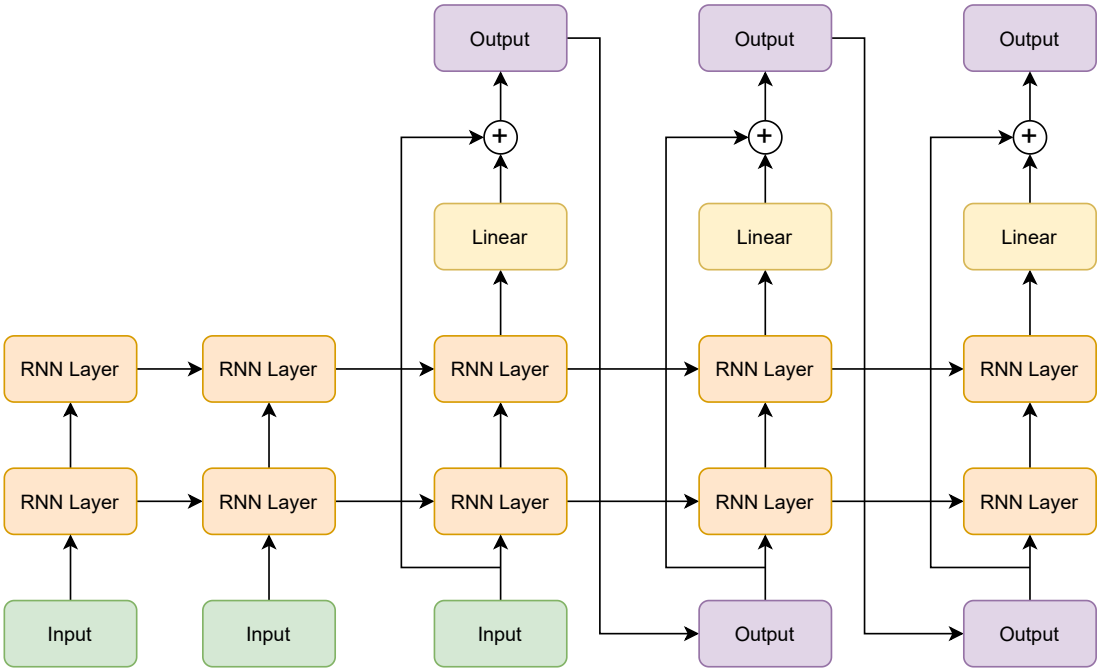


Figure 4.1.: Adopted recurrent architecture. The linear layer predicts offsets to the input. Generated output is used auto-regressively in the next step.

outperformed by the action-aware one. However, action labels may be hard to obtain in practical applications so that we only train action-agnostic versions.

In contrast, we inherit many of their crucial design decisions. Our training follows a sampling-based approach instead of teacher forcing (compare Section 3.1.2) in order to expose the model to its own noise. Furthermore, we employ a residual connection from input to network output so that the model predicts velocities. Martinez et al. motivate that by improved continuity for the very first frame. Lastly, we train the network on all actions jointly to better utilize large datasets.

4.2. Towards Long Predictions

Since we do not use teacher forcing, the network needs to predict latter frames from an extremely noisy estimation of the early frames (see Section 3.1.2). We found it to be helpful to start with a time horizon of just one frame and increasing the time horizon slowly over the course of training. By ensuring that the network can adjust to each time horizon for a few epochs, the network starts with a good initialization when the time horizon is increased by one frame. This has some similarities to the noise schedule Fragkiadaki et al. [FLFM15] employ, because the time horizon of one frame means that the model does not yet see its own output. Unlike their noise schedule, we do not make any assumptions on the

type and magnitude of noise since it solely comes from the network itself once the time horizon contains two frames or more.

4.3. Training

Since we changed the underlying pose representation and added global information to the motion sequences, many of the training details in [MBR17] need to be revised for our purposes. Most importantly, we utilize different loss functions, but some implementation details need to be adapted as well.

4.3.1. Batch Packing

Accumulating samples into a batch matrix requires fixed dimensions which are not given in our setup due to varying lengths of the sequences. The most common solution to the problem is padding, which does not work well with recurrent networks as each input token alters the hidden state. While some RNN variants like GRUs are in theory capable of learning when to ignore input, there is a more elegant solution which [PGM+19] calls packing. Using this technique, the batch dimension is dynamically lowered over time, depending on how many sequences are shorter than the current temporal step. This frees the network of having to learn to differentiate between input and padding.

Batching is not only necessary for the input but for the regression targets as well. Some of the motion sequences are shorter than 2 s so that there are neither enough frames for input nor for output. Ideally, those sequences should be discarded, but that would decrease the amount of multi-person data considerably. Therefore, we still train on these sequences but only supervise those frames where ground truth is available. The missing targets are padded with zeros and the corresponding loss terms are filtered with a binary mask. Training on less frames than the intended number of target frames is also utilized as part of our output length schedule Section 4.2.

4.3.2. Loss Function

A motion prediction problem is a temporal pose regression problem. As such, we can view each time step as an individual pose regression. Therefore, we can consider different pose regression losses. We will be evaluating a squared L2 loss based on Euclidean distance, which resembles the mean squared error angle loss from Martinez et al., and an L1 loss based on Manhattan distance, which is used for some pose regression methods, e.g., in [SLAL21].

Pose Loss

Each joint of a pose is a point in three-dimensional space. Therefore, we can use distance metrics presented in Section 2.1.1 as a measure of how accurate a joint was regressed. Since a pose consists of multiple joints, the loss of a whole pose is the arithmetic mean of distances between ground truth joints and predicted joints. We prefer the mean over a sum because the pose loss becomes invariant to the number of joints and the interpretability is higher, namely a Euclidean pose loss becomes the average distance each joint has to be moved to fit perfectly.

The resulting pose loss using Manhattan distance is given by:

$$L_{pose}(y_{f,p}, \hat{y}_{f,p}) = \frac{1}{J} \sum_{j=1}^{J} \|y_{f,p,j} - \hat{y}_{f,p,j}\|_1$$

where $y_{f,p,j}$ and $\hat{y}_{f,p,j}$ are ground truth and prediction location of joint j respectively at frame f for person p and J is the total number of joints. Furthermore, we denote groups of joints (poses) by $y_{f,p}$, videos of poses by y_p and batches of videos by y, i.e., when grouping along an axis, we leave out the corresponding index.

Motion loss

It is crucial to mask out poses where the ground truth is not known. This may occur for short sequences which are then padded to have the correct length for batching Section 4.3.1. Likewise, it is crucial to divide the sum of pose losses of a single motion by the number of frames in that motion. Not doing that would result in a bias of the model to attend to motions extending over long time intervals. The reason is that those long sequences consist of more frames, therefore the loss becomes larger due to more summands. To summarize, we can say that the motion loss is the arithmetic mean of pose losses of that motion and represents the average pose loss in each frame. It is given by:

$$L_{motion}(y_p, \hat{y}_p, m) = \frac{1}{F} \sum_{f=1}^{F} m_f L_{poses}(y_{f,p}, \hat{y}_{f,p})$$

where m_f is the masking bit of frame f and F is the total number of prediction frames.

Batch Loss

Lastly, we compute the arithmetic mean of all motion losses in a batch to achieve invariance to different batch sizes. These mainly occur because a dual person motion sequence is treated as two independent motion samples so that more dual

person sequences result in larger batches. The advantage of this design decision over an explicit normalization with the number of people is that each prediction is equally important, no matter how many people are present in the scene. The resulting loss is given by:

$$L_{batch}(y, \hat{y}, m) = \frac{1}{P} \sum_{p=1}^{P} L_{motion}(y_p, \hat{y}_p, m)$$

where P is the number of people in the batch.
Alternatively, one might also suggest to combine motion loss and batch loss, i.e., compute the mean pose loss over all frames and all people jointly. However, this would introduce a bias towards long sequences similar to what is explained in Section 4.3.2. This can be illustrated by the following example: let us assume the batch contains two single-person motions, where the first motion consists of 30 frames, the second motion consists of 10 frames and the prediction method has a constant pose loss in each of the 40 frames. Then the first motion would contribute three quarters to the joint loss, despite a constant performance between both sequences.

Absolute and Relative Loss

The loss described in Section 4.3.2 can be achieved in two different representations: in root-relative coordinates and in absolute (camera) coordinates. Both representations encode crucial information. The root-relative loss ensures that the prediction constitutes a plausible human pose irrespective of the human's location with respect to the camera. On the other hand, the absolute loss ensures that all joints move through space as a connected unit. We therefore define an absolute importance weight $\alpha \in [0, 1]$ and choose our final loss as:

$$L(y, \hat{y}, m) = \alpha L_{batch}(y, \hat{y}, m) + (1 - \alpha) L_{batch}(r(y), r(\hat{y}), m)$$

where r is a function which subtracts the root joint location from each frame and each person independently.

4.3.3. Implementation Details

Our RNN models are implemented in PyTorch [PGM+19] and were trained on a single GPU, in most cases a GTX 1080 Ti and in some cases a Titan X. We experiment both with GRUs and LSTMs. Our baseline RNN has two layers and 1024 hidden units. We use Adam with weight decay [LH19] with default parameters as our optimizer and employ gradient clipping in case exploding gradients may occur. All our models were trained for 150 epochs with a batch size of 128, which takes between one and two hours except where we experimented with larger networks. The learning rate starts at 1×10^{-3} and is exponentially decayed by a

factor of 0.98. After 120 epochs, an additional decay by factor 0.1 is applied. Almost all hyperparameters are varied during our extensive hyperparameter study presented in Section 8.2. Notable exceptions are the number of epochs and the learning rate decay which were manually tuned once until a stable setting was found.

5 Forecasting with Transformers

After presenting our adaption of an existing method to global motion forecasting, we will now present our own neural network. It is based on the attention mechanism presented by Vaswani et al. [VSP+17] (see Section 3.1.2) which was recently adopted to computer vision as well, e.g., in [CMS+20]. Our motivation is that attention models all dependencies, e.g., the influence of the first frame on the last frame is considered as much as the influence of two consecutive frames on each other. In contrast, recurrent models have to decide each step how much current information overrides past dependencies. Moreover, each decision is final, i.e., once old information is overridden or new one discarded, it can not be restored. Further details can be found in Section 2.2. One important aspect of attention in our use case is that we can utilize it both for temporal modeling and relationships between multiple interacting humans.

5.1. Architecture

Our architecture is presented in Figure 5.1. It bears many similarities to the encoder of the successful transformer from Vaswani et al. [VSP+17] used for language translation, but tokens represent poses at certain frames instead of words. We will now describe the different components.

The input sequence is embedded with a single linear layer. Afterwards, a positional encoding of the frame indices, consisting of sine and cosine waves with different frequencies, is added to the embedded input in order to provide the model information about the order of frames.
After that, several layers are stacked. Each layer consist of an attention module to aggregate information between different frames. A second attention module aggregates features between poses of different people in order to evaluate whether these poses constraint each other. These modules are followed by a two-layer perceptron with a non-linearity and dropout. Both the attention modules and the

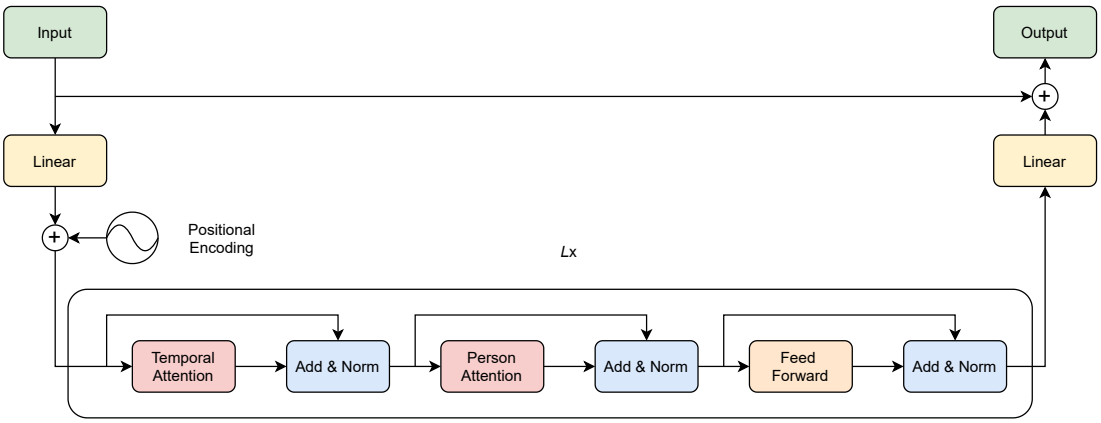


Figure 5.1.: Attention-based motion predictor. Attention is used both between frames and between people in the scene. The model predicts the new pose as joint offsets to the previous pose.

perceptron have a residual connection for better gradient flow. Dropout and layer normalization is applied after each residual connection, refer also to Section 2.2.5. After the last of these layers, the embedded frames are simply projected back into joint coordinates with a linear layer complementary to the very first embedding layer. As is common in motion prediction, the output of the model at each frame is interpreted as a velocity which means that in order to obtain the final poses, the previous pose has to be added to the output of the projection layer.

Many differences between our architecture and the original transformer stem from the differences in the tasks. In this paragraph, we will only list these differences, and refer to Section 3.1.2 for details on how Vaswani et al. implement these aspects and their reasoning. Firstly, Vaswani et al. have a dedicated decoder with different weights to accompany variations between source and target language, whereas our input and output both consist of human poses at different time steps so that new weights are not needed. Furthermore, the usage of attention is adopted to our setting, see Section 5.2. Lastly, Vaswani et al. use auto-regressive sequence generation with beam search, whereas we follow Hernandez et al. [HGM19] (also described in Section 3.1.2) approach to sequence generation, which is the topic of Section 5.3.

Our architecture also took some inspiration from the motion forecasting method in [ACKH20] where attention is used for temporal and spatial reasoning. A key difference is that Aksan et al. consider each joint as an independent input entity whereas we concatenate all joints into a single feature vector of in our case $24 \cdot 3 = 72$ entries which is not considerably more than the 54 dimensions used in [MBR17]. The advantage of concatenation is that it enables the network to argue holistically on whole poses. Another difference is that the method from Aksan et al. is single-person only, i.e., it uses spatial attention between joints of the

same person instead of between poses from different people as ours does. Lastly, their method uses auto-regressive generation like Vaswani et al. (but without the beam search).

5.2. Usage of Attention

The basics of attention are described Section 2.2.4. As shown in Figure 5.1, our architecture utilizes two types of attention. Temporal attention, which is named analogously to [ACKH20], exchanges features between poses of the same person at different points in time. In contrast, person attention aggregates features between poses of different individuals, hence the name. It does so for each point in time independently. In the terminology of Vaswani et al., both our attention modules compute self-attention which means that query, key and value tokens come from the same data distribution, e.g., poses at different points in time. This is opposed to their encoder-decoder attention which is often used to combine data of different modalities, e.g., words from different languages in [VSP+17].

Temporal attention is vital to the success of transformers because they mainly consist of feed-forward layers which consider each frame as an independent input. Only temporal attention allows modeling of temporal dependencies by comparing different frames for similarity and exchanging features based on how similar each frame pair is. In our case, temporal attention models the influence of past poses on future movement. Therefore, attention is the analogue to the internal state of an RNN which also has the purpose to encode knowledge of the past.
Person attention is the module that compares poses of different individuals at the same point in time. Its purpose is to infer constraints from interacting humans, e.g., that the palms touch during a handshake. The huge advantage of attention over other techniques is that it generalizes to an arbitrary number of people. For single-person sequences, person attention essentially becomes another linear layer, i.e., the output is completely determined by the matrix which maps tokens to values. For two people, the attention evaluates how much the second person matters to the feature vector of the first person and vice versa. If person two is not important to the motion of person one, the attention module can ignore the feature vector of person two, i.e., treat it as a single-person sequence. In the opposite case, attention will combine information from both people into the new features. Hence, the attention score is similar to a gate in a GRU which controls how much information is updated. For more than two people, the query-key mechanism searches for pairs which are interacting so that information can be aggregated between those individuals.

Note that person attention and temporal attention are diametrically opposed. Temporal attention looks at each person independently but considers each person's past while person attention only looks at a single frame at each step but

considers all people present in the interaction scenario. It is therefore important to stack multiple transformer layers in order to aggregate information along the missing connection. For example, after the first person attention layer, the feature vector of the pth person contains information of all other $P-1$ people depending on how important each person is for the motion of the pth one. Consequently, when this vector is passed to the temporal attention module of the next layer, the temporal exchange of information is not limited to the pth person. Instead, relevant knowledge from other people is also aggregated to the past of person the p. An alternative way would be to compute full attention between all people at all time steps. However, for P people in F different frames, this would require P^2F^2 comparisons instead of $PF^2 + FP^2$ so that this solution scales badly to larger number of people or frames. Multiple layers would probably be necessary even for combined person time attention so that we ignored that constant for our analysis.

5.3. Sequence Generation

Just like with recurrent neural network, a popular method for transformers to generate sequences is to use auto-regressive formulations. As described in Section 3.1.2, Vaswani et al. consider multiple translations for each word, whereas some authors do not consider such a probabilistic generation approach viable for motion forecasting. But if only one pose is generated at each future time step, we can do that parallel with attention. This is unlike a recurrent neural network, which needs to be sequential in order to update the internal state correctly. We can therefore utilize the inpainting formulation from Hernandez et al. [HGM19] where missing information, i.e., future poses, are replaced with zeros in the input. This is considerably faster for both training and inference. Another advantage concerns temporal attention. Usually, this type of attention needs to be masked in order to stop information leakage from future tokens. Failing to do so would teach the network to look at future tokens in training, which will not work during inference. In contrast, the inpainting formulation already masks future tokens by itself. Consequently, attention masking can be omitted and features can be exchanged forward and backward through time which increases the model's expressive capabilities.
Note that using inpainting does not restrict our method to a certain length since longer sequences can be generated by concatenating more padding symbols after the known history. This is in fact utilized in order to implement the output length schedule mentioned in Section 4.2.

A key difference to Hernandez' formulation is how unknown joints are padded. Hernandez et al. simply multiplied these joints with a binary mask. However, mapping zeros to different joint locations only works for networks with internal state. Although the convolutional network in [HGM19] does not have a state

per se, the convolutions have a limited window size so that the results of the convolution are unique as long as each window has access to a different subset of frames. In contrast, temporal attention compares each query with the same global set of keys. Because zero-padded frames will result in the same queries, the outcome of attention will be deterministic as well so that the method will predict the same pose for all thirty future time steps. Adding the positional encoding to the padded zeros somewhat alleviates the issue but the problem definition still leads to unstable solutions where small disturbances in the input need to be mapped to considerably different outputs. We therefore replaced the padding with a constant velocity heuristic, which applies the difference between the last known frame and the second to last known frame iteratively. In other words, each joint moves along the line defined by the two locations of the joint during the last two frames. Although this heuristic does not produce valid poses, e.g., it violates the bone length constraints, it increases the discriminability between the inputs enough for good prediction results.
Another key difference is the supervision on the known parts. Hernandez et al. [HGM19] use a reconstruction loss on the unmasked joints so that the output is consistent with the input. This makes sense if the network is not only used for motion prediction but also for regressing occluded poses. However, our main emphasis is motion prediction and preliminary tests showed that additionally supervising the historic poses hurts performance, probably because it enforces that past information is represented in a way that allows reconstruction, which may not be optimal for our prediction.

We would like to add a final remark to inpainting and auto-regressive sequence generation. Preliminary tests showed that the former outperforms auto-regressive generation with teacher forcing. Unfortunately, using sampling-based training like [MBR17] is infeasible with the default implementation of multi-head attention due to a lack of caching. In a custom implementation, each future token only requires the computation of one query, one key and one value because all previous queries, keys and values can be reused from the previous auto-regressive step. Due to time constraints, we were not able to implement cached multi-head attention so that we could not evaluate the accuracy of an auto-regressive transformer with sampling-based training.

5.4. Training

Our training setups of transformers and RNNs are vastly similar. Most importantly, the loss function presented in Section 4.3.2 is reused. The differences to RNN training are highlighted below.

5.4.1. Padding and Positional Encoding

Different sequence lengths are padded for the sake of batching. Unlike other approaches, we pad zeros at the beginning of each sequence instead of the end so that the last historic pose is directly followed by the first predicted pose in the positional encoding.
We additionally decided to use negative positional encoding, so that non-negative positions correspond to future poses and negative positions to historic poses. This ensures that those poses most relevant to prediction, namely the last historic poses, have a consistent positional encoding. Future work should empirically investigate whether this is a good choice.
During temporal attention, padded frames should be ignored because they do not contribute meaningful feature vectors. Following Vaswani et al., we add negative infinity to the query-key dot products so that the resulting attention weights will be zero after the softmax operation.
We also apply person padding when working with multi-person datasets. The purpose is solely to store the batch in a tensor with fixed dimensions. The same masking procedure is applied to ignore padding during person attention.
Furthermore, we apply the same target padding and masking procedure as mentioned in Section 4.3.1 in order to utilize short training sequences as well.

5.4.2. Implementation Details

The transformer model was trained on a single Tesla V100 GPU. The default version has eight layers and an embedding space with 128 dimensions. The hidden dimension between the two feed-forward layers has 256 units. The dropout probability is 10 %. We employ eight attention heads for temporal attention and two for person attention. This is owed to the fact that our data contains at most two people so that we hypothesize that more heads are not necessary. Future research should investigate that in combination with a higher variety of the number of individuals. The optimizer is again Adam with weight decay, but with $\beta_2 = 0.98$, i.e., the estimator of the variance of the gradient adapts faster to the current training stage. This was also reported in [VSP+17]. Similar to the RNN, we employ gradient clipping and the same type of loss functions described in Section 4.3.2. We found it helpful to use a smaller batch size of 32. We also reduced the number of epochs to 120 due to resource constraints of non-project jobs on the RWTH Aachen compute cluster. This should not affect the comparability between transformer and RNN because improvements after 100 epochs are marginal for both architectures. The training usually took between one and two hours. The learning rate schedule is the same as in Section 4.3, except that the additional decay by factor 0.1 is applied after 110 epochs instead of after 120 epochs. As for the RNN, we evaluate the performance impact of basically all hyperparameters, see also Section 8.3.

6 Motion Datasets and Processing

This chapter deals with how motion data is obtained, how it can be represented, and what steps are necessary to feed motion data to a neural network.

6.1. Motion Datasets

In order to work with deep learning methods, it is necessary to acquire datasets for training. As discussed in Section 2.1.3, human poses are usually represented by a subset of their joint locations. Obtaining these 3D positions is non-trivial as they lie within the human body and not on the surface. Many approaches to obtaining 3D poses, including professional motion capturing systems, therefore rely on robust triangulation, i.e., 3D reconstruction. This process consists of detecting relevant points in image coordinates across different camera views and then inferring the 3D locations of these points using knowledge about the positioning of the cameras. The points to be triangulated can be markers attached to the cloths of the subjects, e.g., [IPOS14], but some marker-less approaches also detect relevant joints with deep learning and triangulate those [JLT+15].
Unfortunately, this procedure does not always work when multiple subjects interact. This stems from the occlusion that is naturally present in these scenarios. Joo et al. [JLT+15] handle this by applying 480 cameras so that all angles are covered redundantly. However, many of their multi-person scenarios consist of static interactions like discussions. We expect no considerable advantage of multi-person modeling in these cases as constraints mainly arise when subjects touch each other (or are close to). A different approach is chosen in [LHL+17] and [LSP+19] where the Microsoft Kinect v2 sensor is used. The Kinect sensor consists of an RGB video stream and a depth sensor. The 3D reconstructions then relies on video, depth and machine learning. Consequently, the poses are much less accurate as they only rely on a single view. Nevertheless, we will be utilizing these datasets called PKU-MMD [LHL+17] and NTU-RGB+D 120 [LSP+19] for two reasons. Firstly, they show interesting interactions like handshakes, hugging,

kicking, etc., where multi-person modeling is vital to understanding these scenes. Secondly, both datasets rely on multiple Kinect sensors positioned at different angles. We can therefore utilize the different views to gain robustness to occlusion. This process will be described in detail throughout Section 6.4.
We think of the resulting data as a best-effort placeholder for future datasets. For example, Guo et al. [GBXAP21] concurrently worked on such a dataset, which was not available in time for us. Since it is difficult to state definite conclusions of the resulting quality, we additionally evaluate our methods on the AMASS dataset [MGT+19]. AMASS is, to our knowledge, the largest collection of high-quality motion datasets and provides all SMPL parameters to allow for full human body shape reconstruction. Unfortunately, AMASS only contains single-person sequences. Using AMASS serves two purposes. Firstly, it allows us to tune our methods on data which is known to be reliable. More importantly, it also allows us to compare our methods on reliable data in order to exclude that our results are caused by wrong assumptions or biases created during preprocessing of PKU-MMD and NTU-RGB+D 120.

6.2. Pose Representation

We have mentioned in Section 2.1.3 that joint angles allow to encode plausible constraints on the pose more easily. Joint locations on the other hand are easier to integrate into existing 3D modeling frameworks. We will therefore focus on joint locations as input to our baselines and networks. However, there are still many possible ways to encode joint locations.
The most straightforward way is to encode each joint location as a point in the same global coordinate system. The concrete choice of origin and orientation of that coordinate system does not matter for now, although we will later explain why we move the origin to be close to the poses. This representation is easy to process and it also generalizes well to global motion because if the coordinate system is fixed throughout a motion sequence, global motion like walking away from the origin is directly encoded into each joint trajectory.
Another possibility is to encode joints in a relative fashion. This means that for each frame, the origin is moved to the root joint, i.e., the root node of the kinematic tree which defines the order of joints. It is therefore also called root-relative representation. In our case, the root joint is chosen as the pelvis. In other words, for a root-relative representation, the coordinate system follows the pose walking through space. This makes it impossible to reconstruct global motion from the poses alone, i.e., the trajectory of the pelvis with respect to a fixed world point would need to be stored externally. In contrast, this pose representation is bounded so that each joint can only move away from the origin by a maximal distance defined through the sum of bone lengths. Moreover, it disentangles local from global movement, e.g., during walking, the torso is pushed from behind the foot which is anchored on the ground until it is in front of it. Therefore, knee,

ankle and toe of that foot moved backward relative to the torso. In the word coordinate representation, this detail is hard to detect because the foot only moved backwards relative to the torso and not relative to the ground. From now on, we will also refer to poses represented in the same global coordinate system as "absolute" poses in order to juxtapose that representation to the root-relative one. It is important to note that existing literature often removes global rotation, e.g., compare [MBR17, MLS20], which means that the coordinate system does not only follow the human, it also rotates around with the human. On the other hand, we do not remove global rotation from our relative representation in order to make it more comparable with the absolute representation.
By now, we presented one pose description which models global translation on top of global rotation, and one that disentangles global movement from local movement. The next one is called mixed representation and combines the best of both worlds. Each joint except for the pelvis is modeled relative to the pelvis. The root joint in contrast is modeled as offset to an origin fixed over time. As a result, each non-root joint is bounded and describes local movement whereas the root joint describes the global movement. Another way to look at it is that the global trajectory is concatenated to the root-relative pose features. This has the distinctive advantage that all global movement components are enforced to be the same, i.e., when transforming a mixed pose to an absolute pose, each joint is translated to follow the exact same trajectory as the pelvis. In the absolute representation, each joint had its own trajectory and the network needed to ensure that all joints had realistic local movement while also keeping the trajectories to move through space as a connected unit.
The last pose representation we present is a variation of the mixed description. The pelvis is still representing the global pose trajectory, but each other joint is modeled relative to its kinematic parent instead of relative to the root joint. Therefore, each joint apart from the pelvis becomes a vector pointing parallel to the bone inside the limb connecting the two joints, but emerging from the origin. We therefore refer to this representation as bone vectors.
All four representations model global orientation and all except for root-relative can be undone as well. Moreover, any possible transformation between pose representations only consists of subtractions, e.g., between neighboring joints or between root joint and other joints. Therefore, each transformation can be formulated as a matrix multiplication so that it can be applied to whole batches of poses at the same time. It is important to note that only the historic part of a motion is transformed. As mentioned in Section 4.3.2, the loss consists of an absolute component to ensure that the model is learning to predict correct trajectories. Thus, it is most efficient to keep the target sequence in world coordinates.

6.3. From Video to Sample

The lengths of motion videos vary significantly across datasets. However, batching and computational constraints require that training samples should have a similar length. We therefore have to deal with videos which are too long or too short.

6.3.1. Padding Short Videos

We already mentioned in Section 4.3.1 and Section 5.4.1 that we pad short videos with zeros for the transformer and utilize packing to batch different sequence lengths for an RNN. However, in some instances, there is another solution. In contrast to NTU-RGB+D 120, the dataset PKU-MMD contains frames with idling people from before each action technically starts. We can use these to pad the shortest sequences so that the network has more context and learning long-term dependencies might be improved. Furthermore, there is no real performance penalty because the bottleneck is almost completely defined by the longest sequence of the batch. We only apply this padding for up to 1 s of input because more rest poses, i.e., poses of a human standing still, will probably not help the network anyways. The usual padding or packing is then applied to obtain 2 s of input, likewise it is applied on those samples from NTU-RGB+D 120.
This type of padding could technically be done on the target as well, but we decided against that because it would encourage the network to predict rest poses in the long run. This behavior often occurs on difficult actions. Encouraging it further could result in rest pose predictions even for continuous actions like walking.

6.3.2. Trimming Long Videos

Some AMASS sequences are several minutes long. Therefore, trimming these videos is inevitable due to the computational complexity. For consistency, we also applied trimming to Kinect sequences. Trimming the input to 2 s and the output to 1 s is not only common in other methods, e.g., [ACKH20, MLS20], it may also reduce the risk of exploding and vanishing gradients in recurrent neural networks. Moreover, such long sequences are usually either heavily redundant, or actually consist of several unrelated actions so that trimming is a reasonable decision. However, we experiment with different input lengths.

One question is yet to be answered. Given a long video, one has to decide which parts to discard and which to keep. This choice is crucial because without cues of the future movement in the input, the task is inherently ambiguous. Furthermore, if the target contains too many idle poses, the prediction method may learn to always predict a motion that leads to the rest pose as mentioned above. Such a network would be of no interest because it has not learned anything except for

the average pose.
For Kinect sequences, a reasonable approach is to keep the frames in the middle and discard beginning and end. Since most sequences are quite short, the middle often corresponds to the interesting part of a motion where one or multiple joints move a considerable amount. However, this approach is not suited to AMASS, since many AMASS sequences consist of multiple, often independent motions.
To leverage the data from the AMASS dataset, taking random parts of the sequence might work reasonably well. This can also be thought of as an additional data augmentation technique. Random sampling of sub-sequences can also be combined with the previous heuristic by weighting sub-sequences close to the middle of a video higher, similar to a normal distribution. This can efficiently be implemented by sampling the split, i.e., the last frame of the input, as the sum of two uniform variables, similar to how seven is the most likely outcome when throwing two dice at the same time. The massive advantage over rounding a normally distributed random variable is the bounded support, i.e., the random variable will only take values from a certain interval, e.g., between 2 and 12 in the double dice roll example, so that no expensive rejection sampling has to be performed.
There are still a couple of problems with this approach though. First of all, the compromise is suboptimal for both datasets. Moreover, it is not suited well for validation and testing, as the outcome of the random split has a massive impact on the performance. We therefore prefer an exhaustive approach, similar to [MLS20]. This means that every frame whose index is evenly divided by some stride size s is chosen as the beginning of the input, provided there are enough consecutive frames. This heuristic leads to reproducible and fair error metrics and takes full advantage of long motion sequences. It can even be extended to skip training samples which are likely to be irrelevant. We apply such an extension where we only include those samples with sufficient movement during the last frames of input and first frames of target. Consequently, neither samples without any relevant motion nor samples where the relevant motion only starts in the later parts of the target are included in training.

6.4. Preprocessing

Since we are using multiple datasets, it is necessary to perform some normalization steps. Furthermore, the quality of PKU-MMD and NTU-RGBD+D 120, to which we will refer from now on as the Kinect datasets, is sometimes rather low so that we tried to enhance it as much as possible. Note that the vast majority of preprocessing steps described in the following are precomputed and stored to disk to speed up training.

6.4.1. Data Normalization

Some data normalization steps are necessary regardless of the dataset. The very first step ensures that axes are re-ordered to have a consistent coordinate system where the z-axis is the principal axis of the camera and x-axis and y-axis are analogue with the pixel coordinate axis from an imaginary image taken. This step also includes scaling the coordinate system to meters as its unit.
After all other preprocessing steps have finished, it is important to normalize the data. Since motions are represented in world coordinates, it is realistic to assume that a human can move several meters in each direction. However, a motion is completely invariant to translation and the only reason why the the absolute position is important is to infer where the human will be over the course of and after the motion. In other words, the network should ignore the location of a human for everything except its trajectory planning. But internally, a network consists of matrix vector products, and the results will vary a lot between a pose close to the origin and one far away. Consequently, it is best to shift the whole motion so that it evolves around the origin. The most straight-forward solution is to compute the center of mass of all joints and translate the poses so that the center of mass coincides with the origin. Note that special care has to be taken for multi-person scenarios. If every human was centered individually, their absolute locations would be meaningless in relation to each other. But one goal of our work is to aid positioning by aggregating information across interacting humans. Thus, we compute the center of mass across all people and all frames in a training sample.
The final normalization step has a similar purpose but works on a different level. After the previous translation, the action now mainly evolves around the origin. However, there are still massive differences between different features. In a bone vector representation for example, the pelvis location encodes the absolute position whereas each other joint is encoded relative to its parent joint. Thus, the pelvis joint is subject to a high variance, while the toes can only move by a small amount relative to the ankle. We therefore employ a second normalization independently for each feature. As a result, each feature, i.e., each bone vector entry, is a variable with zero mean and unit variance. The mean and variance are computed once over the complete training set and are not updated with test data because these statistics are not known in real inference either. This is unlike the centering operation described above which can still be performed during inference with the help of the known input part of the motion. Note that in the current implementation, the centering is performed on the whole video. Afterwards, feature-wise mean and variance are computed and long videos are split into smaller training samples, as described in Section 6.3.2. A better approach would be to first split long training videos into smaller samples, then center each sample individually based on its input part, and compute feature-wise mean and variance of the training data last. That way, each individual motion sample would

be centered more accurately and centering would be closer to the inference mode as well. Unfortunately, time constraints kept us from realizing that improvement.

6.4.2. Enhancing Multi-Person Data

While the Kinect can easily deal with multiple people in the receptive field, its quality is significantly lower than that of professional motion capturing systems. We therefore performed multiple preprocessing steps to deal with occlusion, challenging poses and missing detections. The preprocessing consists of the following steps:

1. Pose detection with state of the art method
2. Intra-view person identification
3. Inter-view person identification
4. Synchronization between views
5. Calibration of extrinsic camera parameters
6. Filtering of implausible poses
7. Fusion of the different views
8. Filtering of implausible poses
9. Aligning ground plane with xz-plane (only NTU-RGB+D 120)

Steps 1, 2 and 7 were done in close correspondence with Sárándi [Sár21]. Many of the design decisions were based on what qualitative looks good as we discovered that quantifiable metrics poorly correlated with what works visually. Nevertheless, we will try to motivate our preprocessing pipeline in the following paragraphs.

Detection and Identification

As the first step suggests, we do not utilize the Kinect detections directly as it tends to struggle with challenging or occlusion-heavy actions like putting on a jacket or hugging another person. We therefore employ MeTRAbs [SLAL21] as a detector in order to regress joint locations in world coordinates on a metric scale.

Next, it is necessary to eliminate false detections and track poses across frames within each Kinect view in the case of dual-person interactions. This is solved with the method presented by Wojke et al. [WB18] which was trained on the MARS dataset for person re-identification [ZBS+16]. The resulting tracks have a high quality but occasionally fail whenever the pose detector only regresses one

of the two pose or the same pose twice, which mainly happens during the aforementioned hugging scenes where both people are closely interacting. Therefore, we first pad the last known pose in those frames with only one detection, which only occurs on NTU-RGB+D 120, so that each frame in a dual-person sequence has two poses where one may either be padded from a past frame or a duplicate of the other pose of the same frame. Afterwards, we apply a Hungarian matching of the pelvis locations at each later frame with their locations at the first frame. This works well on these datasets because the actors typically stay close to their original position and, in case of PKU-MMD, even return to that position after each interaction. Consequently, the real poses are now correctly grouped to tracks for the vast majority of dual-person sequences, although tracks may also contain padded poses. Note that the re-identification step is necessary regardless of the Hungarian matching, namely in order to clean up the tracks from false positives. Otherwise, these detections would interfere with the Hungarian matching and lead to poor tracks.

The next step matches people across different views. Since the poses withing each view are already matched correctly, this simplifies to a binary decision between matching the first person of one view to the first or the second person of the other view. The main problem of this step is that the different views are not yet synchronized well, especially for PKU-MMD sequences, but synchronization is difficult without knowing correct pose correspondences. This is why we first solve the slightly simpler problem of intra-view tracking before we match across different Kinect cameras. The method used for inter-view matching is applying synchronization on both possible matches and choosing the one with lower error. The synchronization method is described next.

Synchronization

Synchronizing the videos is quite difficult for PKU-MMD sequences. Here, multiple interactions are recorded in one video. Unfortunately, the Kinect RGB video, on which MeTRAbs depends, can record at 15 Hz or at 30 Hz based on lighting. Qualitative tests confirmed that a constant frame offset is not sufficient to obtain good synchronization. Even worse, the only time stamps Liu et al. provide are those that mark the start and end of each action. These labels are manually created and are therefore occasionally wrongly ordered, highly inaccurate or incomplete. Nevertheless, we found it to be helpful to use these time stamps whenever available, i.e., we apply synchronization on sub-sequences defined by those time stamps. The algorithm used for synchronization is presented in Section 2.3.2. A difficulty at this stage is that the cameras utilize different world coordinate systems so that one cannot compare joint locations directly, but estimation the Euclidean transformation between to cameras is extremely difficult without synchronization. As a solution, we transform the joint locations to an-

gles between limbs since angles are preserved by Euclidean transformations such as camera movement. Note that we explicitly do not refer to joint angles, as joint angles are in fact rotations with three degrees of freedom and require the definition of appropriate local coordinate systems at each joint in order to define these rotations unambiguously. Instead, we compute the angle defined by the three points of parent joint, current joint and child joint. This representation of a pose is ambiguous and only used for synchronization. The metric used to compare two poses is the mean absolute distance clipped to a maximal threshold for robustness against outliers:

$$d(x,y) = \frac{1}{J}\sum_{j=1}^{J} \min\left\{\frac{|x_j - y_j|}{T}, 1\right\}$$

where J is the number of angles, T the outlier threshold, and x_j and y_j refer to the jth angle of pose x and pose y respectively. Note that d is a proper distance metric. The function d is positive definite and symmetric, which directly follows from $d_a(x,y) = |x-y|$ being a distance function. For triangular inequality, consider

$$\begin{aligned}
\min\left\{\frac{|x_j - y_j|}{T}, 1\right\} &= \min\left\{\frac{|x_j - z_j + z_j - y_j|}{T}, 1\right\} \\
&\leq \min\left\{\frac{|x_j - z_j|}{T} + \frac{|z_j - y_j|}{T}, 1\right\} \\
&\leq \min\left\{\frac{|x_j - z_j|}{T}, 1\right\} + \min\left\{\frac{|z_j - y_j|}{T}, 1\right\}
\end{aligned}$$

Since the triangular inequality holds for each summand, it also holds for the whole sum, therefore we conclude that $d(x,y)$ is a proper distance metric.

Another difficulty during synchronization is that we want to align three views with each other, but common implementations which are not tailored towards ,e.g., DNA sequences only align two time series. Feng et al. [FD87] devised an easy generalization of pairwise alignment algorithms by aligning multiple sequences progressively, starting with the most similar ones. We adopted this approach to our synchronization problem. After the initial alignment, we discard frames in both sequences such that each frame only aligns to a unique frame in the other sequence, i.e., until there exists a bijection between the frame indices. Then, this procedure of dynamic time warping and deleting is repeated to align the pair with the last camera view.
Alternatively to deleting frames, one might also duplicate frames until a bijection exists. Both strategies have their drawbacks. On the one hand, deleting frames can lead to rather abrupt motions which may hinder convergence and are not really suitable for evaluation either. On the other hand, duplicating frames leads to slow motions and encourages the network to predict static poses without any

movement over time. Since predicting a non-moving poses is neither difficult nor interesting, we decided for deletion.

Calibration, Filtering and Fusion

Once the synchronization is performed, we can now proceed to determine the extrinsic camera parameters. This requires estimating a rotation matrix for adjusting the viewing angle and a translation vector for moving the camera position. The translation can be computed from the two center of masses. In order to compute the special orthogonal matrix rotating between the two centered point clouds, we employ Kabsch' algorithm wrapped into RANSAC, compare Section 2.3.3. Outlier robustness is necessary due to duplicate detections, padding and other noisy poses. We consider a point pair to support a transformation hypothesis if the observed point lies within a 5 cm radius of the transformed point. We always sample 500 transformations, i.e., without early stopping, and choose the transform with the highest number of inliers. Even if 75 % of all joint pairs contained an outliers, the chances of a wrong hypothesis are only

$$(1 - 0.25^3)^{500} \approx 0.038\%$$

since only three joint pairs are required, compare [HZ03].

Afterwards, we filter poses where the bone length deviates significantly from the average bone length of each dataset. In order to consider a pose implausible, the bone length must deviate at least 15 cm from the average bone length. Furthermore, it must either be twice as long or half as long as the average. Both conditions are insufficient on their own. Large bones such as the thigh bone can naturally vary between a small and a large human so that just checking for the absolute difference is not sufficient. Likewise, some joints like the hands are usually quite noisy so that a relative difference of factor 2 is not that problematic. Note that we do not filter poses which were padded from a previous frame yet, since this padding has acceptable quality for a few frames so it should be kept for the fusion step.
This step could have been performed earlier in the pipeline as well. Since the camera calibration benefits from RANSAC regardless, it does not really matter if the calibration is performed before or after the filtering step. It is rather unclear whether synchronization would work better after filtering. On the one hand, using extremely noisy poses for synchronization might lead to bad decisions even though we employ an outlier-robust distance metric. On the other hand, deleting frames before synchronization means that many plausible or even optimal alignments become impossible so that the synchronization produces worse results.

As mentioned before, the next step is the fusion of the poses after they have been transformed to the same coordinate system. This is done with the help of

the geometric median, which is more robust against outliers than the center of mass. The geometric median is computed for each joint separately, so that the resulting pose closely sticks to the more similar pair between the three views. The resulting pose is therefore quite robust as long as at least two of the three views are not corrupt.
Since there does not exist a closed-form solution, we use the algorithm presented in [VZ00].

Final Filtering

After the fusion, we filter those frames where a pose is static over multiple frames, where a pose lies outside the expected range, i.e., more than 10 meters away from the origin, or the where the bone length check mentioned in Section 6.4.2 fails. These cases still occasionally occur after the fusion if the majority of poses was corrupt. However, most motion sequences require minimal or no filtering at all. Unfortunately, some sequences, especially from NTU-RGB+D 120, contain a lot of frames deleted by the filtering. This may result in serious discontinuities during the motions. We discard sequences with extreme discontinuities from the validation and test split, compare Section 7.2.2.
One final step is performed for NTU-RGB+D 120. Here, motion sequences are grouped into set-ups and each set-up has a different placement of the Kinect sensors. Sometimes, they are placed at an angle so that the ground plane is not parallel to the xz-plane, which we want to correct in order to analyze global motion prediction in a more basic form. We therefore collect videos which show actions related to walking and extract the toe joint locations. By restricting the collection to walking-related actions, we ensure that the points are not almost collinear in which case the estimation becomes sensible to outliers. We furthermore perform the ground plane normalization set-up wise to have a larger sample size for estimation. After we collected a sufficient number of toe locations and centered the point cloud, we project these points onto the xz-plane by setting the y-coordinate to zero. After rescaling the vectors to their original length, we can apply Kabsch algorithm [Kab78] to compute a rotation between the xz-plane and the oblique plane containing the toe joints.

6.4.3. AMASS Dataset

Since AMASS consists of videos with high-quality data from motion capturing systems, the preprocessing only consists of two steps. The first one is to apply forward kinematics to the axis-angle representation in order to obtain joint locations in world coordinates.
The other step is the normalization of frame rates. The frame rates of the motion data in AMASS varies between 250 Hz and 60 Hz. In order to have meaningful state transitions in an RNN and a meaningful positional encoding for transformer, the frame rate should be more or less the same for all training samples.

The easiest solution is to take equidistant frames from all sequences, e.g., if every 25th and every 6th frame are taken respectively, both sequences would have a frame rate of 10 Hz. Unfortunately, such a frame rate would have insufficient quality and 10 is already the greatest common divisor of 250 and 60, so that each larger divisor leads to non-integer stride sizes for at least on type of frame rate. Mao et al. [MLS20] address this by rounding down fractional stride sizes to the next integer. Since their target frame rate is 25 Hz, they therefore sample the low-frequency sequences with a stride size of $\left\lfloor \frac{60}{25} \right\rfloor = 2$. Consequently, they are downsampled to 30 Hz, which means that when predicting 25 frames on a low-frequency sequence, in reality only approximately 0.83 s are predicted. We therefore use the same method but with a target frame rate of 30 Hz so that at most 0.1 s are missing on those samples with a frame rate of 100 Hz.
A more elaborated solution would be to not round the stride size but the indices to sample, e.g., sample frames 0, 1, 3, 4, etc., for a stride size of 1.3. However, one can already see that the resulting frame rate varies by factor 2 within such a sequence which is undesirable as well. Another possibility when using non-integer stride sizes is to interpolate between the two neighboring frames, but this degrades the data quality so we decided for the much simpler heuristic used by Mao et al. [MLS20].

6.5. Data Augmentation

Now that we described the preprocessing, we can look at data augmentation. This technique is designed to reduce overfitting and diversify the datasets by modifying each motion so that the network never sees the exact data sequence twice. To that end, we implemented transformations which are expected to occur for poses detected in the real world as well.
Our first augmentation is a random rotation. We rotate around the y-axis so that the ground plane stays horizontal. A random rotation angle is sampled, the according rotation matrix is constructed and the whole video is rotated by the resulting matrix. This provides robustness against different camera angles on the same scene.
The second augmentation is a simple scaling operation, i.e., all poses are scaled by a random factor between 0.8 and 1.2 since humans can naturally vary in size. The correct way to achieve this would be to transform the poses to bone vectors and then scale theses bone vectors. However, the augmentation should be applied to all types of input representation. A cheap approximation is to just multiply the joint location by the factor. This will stretch some bones more than others which may provide additional robustness.
The last augmentation is flipping, which is applied with a probability of 50 %. Most actions can be performed by the right hand as much as the left one. Flipping consists of two steps. The first one is reflection, which can for example be achieved by multiplying the x-coordinate with -1. The other step is to swap right and

left joints. To see why this is necessary, consider a person standing at the origin and stretching its right arm to its right side. Reflection will result in the right arm pointing to its left with the person still looking into the same direction. But right should and left shoulder cannot exchange their position without the person turning around by 180°. Therefore, exchanging right and left labels lead to the left arm pointing to the left, which is what was desired. In terms of implementation, this step is implemented as a fixed permutation of the joint order. Note that the permutation of joints depends on the pose representation so it is best to apply augmentations before pose transformation to only require a single implementation.

7 Evaluation

In order to compare the transformer architecture with the adopted recurrent network and state of the art methods, we utilize different metrics in order to analyze their performances from different angles. We will therefore first define these metrics and compare how they work. Afterwards, we are going to present the evaluation protocol, which is adopted from [MLS20] for maximum comparability. We also introduce a new protocol for multi-person evaluation which we will motivate thoroughly.

7.1. Metrics

Since all metrics are computed in a similar way, we will first explain the general procedure before defining them mathematically.
On Human3.6M, metrics are evaluated on each action individually, as difficulty and performance greatly varies between simple actions like walking and complicated ones like greeting (compare [MBR17]). Unfortunately, AMASS consists of a wide variety of different datasets and therefore has a large amount of different actions, similar to NTU-RGB+D 120, which was originally designed for action recognition. Thus, for the sake of overview, we will evaluate all actions jointly although we will also describe how one would proceed otherwise.

The procedure is similar to how the loss is computed in Section 4.3.2 and summarized in the following equation:

$$M^{(f)} = \frac{1}{P_f} \sum_{p=1}^{P_f} \frac{1}{J} \sum_{j=1}^{J} M(y_{f,p,j}, \hat{y}_{f,p,j}) \quad \forall f \in \{1, \dots, 30\}$$

For each frame and each person, a pose error metric $\sum_{j=1}^{J} M(y_{f,p,j}, \hat{y}_{f,p,j})$ is computed similar to the pose loss in Section 4.3.2. The concrete formula for M depends on the metric, but all metrics are averaged over the number of joints J

similar to Section 4.3.2.
Unlike the scalar loss in deep learning, analyzing the performance at different time horizons might be interesting so the aggregation over time is postponed to a later stage of the metric computation. Instead, after computing the per-pose metric, the next step is to compute the analogue of the batch loss. This means, that for each time step the sum of all per-pose metrics of that action type is computed, and divided by the number of occurrences P_f of that action type. Two special cases might occur here. Firstly, some samples of that action type might have less than 30 frames and therefore less than 30 per-pose metrics. In that case, the afflicted sample does not contribute to the error metric at the later time steps, which is the reason why P_f might not be the same for different frames f. The second special case occurs with dual-person actions. Here, each person contributes an individual per-pose metric, i.e., the number of per-pose metrics is two times the number of such dual-person videos. For our data, this is equivalent to explicitly averaging over the number of people, because the number of people is constant for each action. The equality can easily be verified by applying the distributive law. Whenever we evaluate all action types jointly, we considered them to be of the same action type. Consequently, the error metric at the first prediction step is then the arithmetic mean of the entire test set at that prediction step and P_1 becomes the test set size.
At this stage, there are 30 metrics M_f per action (or in total when computing metrics jointly) describing different time horizons. The next step is the temporal aggregation. Since it is interesting to evaluate the network for different tasks, the metrics at different time steps are relevant. A common distinction (compare [MBR17], [MLS20]) is between short-term motion forecasting, which measures the performance after 400 ms (12 frames at 30 Hz), and long-term motion forecasting, which measures the performance after 1 s (30 frames). Longer time horizons are sometimes used as well [KBM+20, GSAH17], however, those require less prevalent evaluation techniques because of the inherent ambiguity of the task over such time horizons. Since this is not the goal of this thesis, we concentrate on the aforementioned time horizons of 12 and 30 frames. Instead of computing the average performance over the first 12 frames, another possibility is to simply take the performance at frame 12. This works just as well because of how correlated the prediction at frame 12 is with all previous time steps. If the prediction in previous frames is highly inaccurate, the performance at that frame will most likely be inaccurate as well. Indeed, we never encountered a case where the error was not monotonically increasing with time. Since the prediction at the twelfth frame is by far the hardest of the first 12 predictions, using the performance at frame 12 amounts to a worst-case analysis, whereas the average performance over all 12 frames amounts to an average-case analysis. We will switch between both approaches depending on the concrete evaluation goal.

Note that the entire evaluation protocol described above is consistent with the

literature [AKH19, MLS20]. This is crucial for a meaningful comparison. Since the evaluation of short sequences and dual-person sequences is to the best of our knowledge not yet consistent in the literature, we integrated those special cases in what seemed to be the most consistent way to us.

7.1.1. Definitions

Let $f \in \{1, \ldots, 30\}$ denote the index of the 30 predicted frames. Let $P_f \in \mathbb{N}$ be the number of people performing a certain action type and let $J \in \mathbb{N}$ denote the number of joints in the underlying skeletal representation of human body poses. Lastly, let $\hat{y}_{f,p,j}$ denote the estimated location of joint j from person p in frame f and $y_{f,p,j}$ the corresponding ground truth location.
The percentage of correct keypoints (PCK) [MRC+17] is defined as:

$$M_{pck}^{(f)}(\tau) = \frac{1}{P_f} \sum_{p=1}^{P_f} \frac{|\{\hat{y}_{f,p,j} \mid \|y_{f,p,j} - \hat{y}_{f,p,j}\|_2 \leq \tau\}|}{J}$$

Here, τ denotes the threshold describing the accuracy required to consider a predicted joint as correct.

The mean per joint positioning error (MPJPE) [IPOS14] is defined as:

$$M_{mpjpe}^{(f)} = \frac{1}{P_f} \sum_{p=1}^{P_f} \frac{1}{J} \sum_{j=1}^{J} \|y_{f,p,j} - \hat{y}_{f,p,j}\|_2$$

The area under the PCK curve (PCK AUC) [MRC+17] is defined as:

$$M_{auc}^{(f)}(T) = \frac{1}{T} \int_0^T M_{pck}^{(f)}(\tau)\, d\tau$$

Sárándi et al. [Sár21] showed that PCK AUC has the closed-form solution:

$$M_{auc}^{(f)}(T) = \frac{1}{P_f} \sum_{p=1}^{P_f} \frac{1}{J} \sum_{j=1}^{J} \max\left\{0, 1 - \frac{\|y_{f,p,j} - \hat{y}_{f,p,j}\|_2}{T}\right\}$$

Note that for all metrics, it does not matter whether we explicitly normalize over the number of people n in a multi-person scene, treat them as nJ instead of J joints or treat them as nP instead of P independent samples:

$$\frac{1}{P} \sum_{p} \sum_{i=1}^{n} \frac{1}{nJ} \sum_{j} x_{p,n,j} = \frac{1}{P} \sum_{p} \frac{1}{n} \sum_{i=1}^{n} \frac{1}{J} \sum_{j} x_{p,n,j} = \frac{1}{nP} \sum_{p} \sum_{i=1}^{n} \frac{1}{J} \sum_{j} x_{p,n,j}$$

which follows from the distributive law. We chose the last possibility and implicitly assume that $P_f = nP$ where n is the number of subjects in that action type. This equality only holds for a constant number of people in each action, which works out with the chosen datasets. However, when evaluating all actions jointly, this assumption is violated. For example, consider a test set with one single-person sequence and one dual-person sequence. For simplicity, assume that the single-person sequence results in an MPJPE of 1 mm and each person in the dual-person sequences is predicted with an error of 2 mm. Explicitly normalizing over the number of people results in $\frac{1}{2}(\frac{1}{1} + \frac{2+2}{2}) = \frac{3}{2}$, whereas considering the dual-person sequence as two independent samples results in $\frac{1}{3}(1 + 2 + 2) = \frac{5}{3}$. We can see that explicitly normalizing over the number of people leads to a final error closer to that of the single-person sequence. In other words, the importance of the two dual-person predictions is diminished because they are logically grouped into one motion sample. This does not make sense for our analysis as the goal is to properly model human motion with multiple people involved. Therefore, we will not group dual-person sequences into one sample so that each generated future motion sequence has the same importance regardless of the number of people involved in the original interaction. Consequently, we assume in the aforementioned metric definitions that $P_f = P_s + 2P_d$ for P_s single-person videos and P_d dual-person videos in the test set.

7.1.2. Differences Between the Metrics

The mean per joint positioning error at frame f is, as the name suggests, the average euclidean distance between each predicted joint and its ground truth location at frame f. It is usually computed in millimeters [IPOS14] and higher scores denote a worst prediction quality. The metric is quite susceptible to outliers, i.e., the MPJPE can become arbitrary large due to a single prediction error caused by corrupt data.

The percentage of correct keypoints works quite differently. A predicted joint is considered correct if and only if it lies within a certain ball around the ground truth location so that accurate models will obtain a higher percentage. The exact distance does not matter, which is reasonable considering that data is never completely accurate and that for example a hand is in reality not a zero-dimensional point in space. Likewise, it does not matter if a human is located 1 m or 2 m away from the prediction, since in both cases, the prediction system failed completely. However, the hard decision threshold between correct and wrong prediction is of course arbitrary. On the one hand, it needs to be chosen large enough so that one can say that a higher error is equally wrong no matter how much larger it is. On the other hand, choosing the threshold higher means that one cannot properly differentiate between a highly accurate model and one that is always wrong by a small amount. Since MPJPE does not have this problem, ties in PCK should be

solved by comparing the mean per joint positioning error to see if one model is systematically more accurate.

The PCK AUC is normalized to a value between $[0, 1]$. If the normalized area under the PCK curve becomes 1 in the interval $\tau \in [0, T]$, it means that even under the most rigorous PCK threshold, all joints are considered to be correct. A lower PCK AUC means that only more relaxed PCK thresholds result in saturated PCK scores, e.g., only at a threshold of 10 cm or more are most joints predicted correctly.
In order to compare PCK AUC to the other metrics, let us first reformulate it. To this end, define $d_{f,p,j} = \|y_{f,p,j} - \hat{y}_{f,p,j}\|_2$. As mentioned in Section 7.1.1, the PCK AUC can be computed with:

$$M_{auc}^{(f)}(T) = \frac{1}{P_f} \sum_{p=1}^{P_f} \frac{1}{J} \sum_{j=1}^{J} \max \left\{ 0, 1 - \frac{d_{f,p,j}}{T} \right\}$$

Rewriting the expression as:

$$\begin{aligned} M_{auc}^{(f)}(T) &= \frac{1}{P_f} \sum_{p=1}^{P_f} \frac{1}{J} \sum_{\substack{j=1, \\ d_{f,p,j} \leq T}}^{J} \left(1 - \frac{d_{f,p,j}}{T} \right) \\ &= \frac{|\{(p, j) \mid d_{f,p,j} \leq T\}|}{P_f J} - \frac{1}{T} \left(\frac{1}{P_f} \sum_{p=1}^{P_f} \frac{1}{J} \sum_{\substack{j=1, \\ d_{f,p,j} \leq T}}^{J} d_{f,p,j} \right) \end{aligned}$$

We can see that the term in brackets is the MPJPE where outliers, i.e., errors larger than T, are ignored. The other two terms then transform the range into the interval $[0, 1]$. Note that the transform would almost be linear if the fraction was not the percentage of outliers. In summary, the PCK AUC is the percentage of inliers minus the MPJPE on the inliers, where the scale of the MPJPE is defined by the threshold T. It can therefore be viewed as a middle ground between MPJPE and PCK showing fine-grained differences comparably to MPJPE but being robust to outliers similar to PCK.

7.2. Evaluation Protocols

We will use three evaluation protocols: One for hyperparameter studies, to which we will refer as ablation protocol, one for evaluating multi-person performance and one for comparison to existing literature. In general, all three protocols are similar, but some of the settings in the literature protocol may not be optimal

for multi-person evaluation.

As is common in motion modeling, we report the performance of the models after 0.4 s and after 1 s of forecasting. Note that unlike some previous authors like Martinez et al. [MBR17] we do not specifically train our network for shorter time horizons, i.e., all scores are achieved by models trained on 1 s of forecasting. Preliminary tests showed that training on short-term forecasting improved the MPJPE by 5 mm after 0.4 s, while increasing the error by roughly 7 mm after 1 s. Thus, the overall performance is really similar for both time horizons, which is why we simplify the evaluation procedure and only consider long-term forecasting models at the cost of slightly higher scores after 400 ms.

7.2.1. Comparison to Literature

To our knowledge only [MLS20, ACKH20, AKH19] evaluated on the AMASS dataset. However, their are two problems with the evaluation from [ACKH20, AKH19]: firstly, they use a preliminary version of AMASS because the full version was not yet available. Secondly, they only use 15 of the 24 joints for evaluation. While most of the literature removes global motion by translating the origin to the pelvis joint, Aksan et al. also ignore wrists, hands, ankles and toes. These are by far the most difficult joints to predict because they have naturally the highest variance in location and also suffer from error propagation if a method uses local joint coordinate systems. In contrast, Mao et al. [MLS20] ignore the pelvis (due to root-relative setting), hips, the spine joint at the height of the belly and the hand joints. The joints adjacent to the pelvis are presumably ignored because they become trivial in a root-relative evaluation. The hand joint locations are tightly correlated with the wrist joint locations so that the network still has to learn their movement (although the final metrics put less emphasis on them because a wrongly predicted arm configuration is only penalized through one joint instead of two). Therefore, we will focus on the comparison to Mao et al., who only report MPJPE. Additionally, we will provide PCK AUC with a threshold of 110 ms analogue to Aksan et al., but evaluated over the difficult joints as well. Note that the train-validation-test split from Mao et al. results in a validation split which is significantly more difficult than the test split. Their method obtains 67 mm MPJPE during testing but 112 mm error during ablation which is backed up by our own results.

7.2.2. Differences of the Evaluation Protocols

In Table 7.1, we will see a direct comparison of the evaluation protocols. The literature protocol is identical to [MLS20], and we will now explain why we deviated from that protocol for ablation and multi-person evaluation.

setting	ablation	literature	multi-person
datast	Kinect / AMASS	AMASS	Kinect
split	validation	testing	testing
use short sequences	yes	no	yes
use idle sequences	yes	yes	yes
skip rate	30	5	5
augmentation / shuffle	no	no	no
Kinect filter threshold	0.4 m	-	0.4 m
input length	varies	2 s	2 s
output length	1 s	1 s	1 s
frame rate of AMASS	30 Hz	25 Hz	-
mean over time	yes	no	yes
less joints	no	yes	no
keep global orientation	yes	no	yes
absolute and relative	yes	yes	yes
short term and long term	yes	yes	yes

Table 7.1.: Differences in the evaluation protocols.

Unlike Mao et al., we also evaluate on short sequences with the help of reduced input lengths and masking if less than 1 s is available for output. The reason lies in the Kinect datasets which consist of a large number of short sequences. This also affects how we evaluate: Mao et al. evaluate the performance after the duration, i.e., at the fth frame. However, we evaluate the performance until each time step, i.e., the average performance over time. The reason is that if a sequence is so short that it only yields 29 of the required 30 frames of future ground truth, an evaluation at the 30th frame would completely ignore this sequence because the performance at the 30th frame is the performance averaged over all sequences with 30 frames available. This is avoided by the temporal mean.
The skip rate defines how many samples are extracted from long videos, e.g., a rate of five means that each fifth frame becomes the first frame of ground truth. We use higher skip rates during tuning to reduce execution time. Furthermore, we experiment with different input lengths during ablation unlike most related research. Another difference comes from the number of joints which are evaluated. As mentioned before, Mao et al. [MLS20] ignore some joints trivial in a root-relative setting as well as the hand joints. Since we want to model absolute positions and see how multi-person modeling can improve the predictions, the hand joints are crucial for our tests, as they are probably the most interactive joint in a multi-person scenario.

As mentioned several times, existing literature typically removes global rotation and position. In order to make the comparison to state of the art fair, we use Kabsch' algorithm (see Section 2.3.3) to remove the rotation between predicted and target pose but only in the literature protocol. We provide the performance when alignment is computed over the minimal amount of three joints and over all joints. The former is a lower bound since our network is not aware during training that a subset of joints is used for rotation. The latter approach is an upper bound because the best possible alignment is used.
The last difference is the frame rate to which AMASS is downsampled to. As discussed in Section 6.4.3, a frame rate of 30 Hz leads to a time horizon closer to 1 s.

As mentioned in Section 6.4.2, our preprocessing on PKU-MMD and NTU-RGB+D 120 occasionally results in large discontinuities. We filter sequences where joints move more than a threshold between consecutive frames even in evaluation. In Table 7.2, we can see that 0.4 m retains more than 90 % of the test data while filtering at a reasonable high velocity of $43\,\mathrm{km\,h^{-1}}$. AMASS data quality is high enough to omit this step.

threshold (m)	velocity ($\mathrm{km\,h^{-1}}$)	data kept (%)
10	1080	100
1	108	99.3
0.5	54	93.6
0.4	43	92.3
0.3	32	88.5
0.2	22	74.5
0.1	11	32.0

Table 7.2.: Effect of different filtering thresholds for Kinect data.

8 Results on AMASS Dataset

After having presented the evaluation protocols used, it is now time to analyze the results. This chapter focuses on high quality single-person data from the AMASS dataset, while in Chapter 9, we investigate the performance on multi-person data. Here, we will evaluate the impact of important hyperparameters on LSTMs, GRUs and transformers. The less important hyperparameter analyses can be found in Appendix A. After all hyperparameters are tuned, we evaluate the performances of our models on test data from AMASS. The evaluation consists of a quantitative comparison to state of the art and a qualitative analysis.

8.1. LSTM Tuning

The first hyperparameter analysis aims at finding a suitable LSTM architecture. We will look for an optimal number of layers and then compare the performance with a GRU in order to decide which RNN design should be analyzed in more details.

8.1.1. Number of Layers

Martinez et al. [MBR17] used a single-layer GRU, but AMASS is considerably larger and more difficult than Human3.6M. Therefore, we will investigate how deep the RNN should be, i.e., how many identical recurrent layers need be stacked for optimal performance.
In Table 8.1, we can clearly see that two layers are optimal. It allows a considerable improvement of 4 mm over just having a single recurrent layer while more layers do not yield a performance gain above statistical variation. Table 8.2 reinforces the point that two layers and three layers lead to the same performance as the PCK scores equal over the time of 1 s. Since adding a third layer requires 30 additional computation steps due to the temporal dependency, it is definitely not recommended to use more than two layers.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
1 layer	53.5	124.2	86.2	61.8	0.635	0.405
2 layers	**49.9**	**119.7**	**87.4**	**63.2**	**0.655**	**0.422**
3 layers	50.6	120.2	87.2	63.0	0.651	0.420

Table 8.1.: Absolute error metrics for LSTM with different number of layers.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
1 layer	42.3	82.5	89.5	75.6	0.703	0.543
2 layers	**40.0**	**80.3**	**90.2**	**76.4**	**0.716**	**0.555**
3 layers	40.6	80.8	90.1	**76.4**	0.713	0.553

Table 8.2.: Relative error metrics for LSTM with different number of layers.

8.1.2. LSTM and GRU

Next, we are going to compare the performance of an LSTM to the simplified GRU architecture. All parameters including the dimension of the hidden state are chosen to be identical for maximum comparability. Note that it is hard to decide whether the hidden state size should be equal during this comparison. The internal state of an LSTM consists of the cell state and the output at the previous time step, therefore it has double the size of the GRU hidden state. On the other hand, only the cell state is protected by forget and input gate so that the previous hidden state cannot be considered as an extension to the memory of the RNN cell.
Table 8.3 and Table 8.4 shows that the GRU architecture completely outperforms LSTMs on the validation split both in global tracking and in relative pose quality. Note that this backs up [MBR17]. We will therefore focus on tuning GRUs for the rest of this work.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
LSTM	49.9	119.7	87.4	63.2	0.655	0.422
GRU	**41.7**	**104.8**	**90.5**	**67.0**	**0.694**	**0.451**

Table 8.3.: Absolute error metrics for LSTM and GRU.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
LSTM	40.0	80.3	90.2	76.4	0.716	0.555
GRU	**36.1**	**75.0**	**91.9**	**78.2**	**0.736**	**0.568**

Table 8.4.: Relative error metrics for LSTM and GRU.

8.2. GRU Tuning

Since GRUs work better on the validation split, it is now necessary to investigate the other GRU parameters for choosing optimal values.

8.2.1. Number of Layers

Comparably to Section 8.1.1, we start with varying the number of stacked recurrent layers. In Table 8.5, we can clearly see that one GRU layer, in contrast to one LSTM layer, is not enough as the model clearly underfits. However, the choice between two and three layers is not so trivial. We can observe a clear increase in performance by moving from two layers to three. However, the resulting network is almost 50 % larger and likewise, the training duration increased by roughly 50 % from 92 min to 136 min. Thus, we conclude that the additional overhead does not justify the rather small increase in accuracy and therefore continue our experiments with two layers. Using three layers for an application is however still viable if accuracy is of uttermost importance.
The relative pose quality visualized in Table 8.6 shows an identical behavior so that we know that one layer not only fails to model global movement but also to produce realistic poses.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
1 layer	91.7	173.3	72.3	48.6	0.451	0.280
2 layers	41.7	104.8	90.5	67.0	0.694	0.451
3 layers	**40.0**	**102.6**	**91.0**	**67.8**	**0.705**	**0.460**

Table 8.5.: Absolute error metrics for GRU with different number of layers.

8.2.2. Dropout

Regularization techniques like dropout are crucial to deeper networks. In our case however, dropout does not help to produce more realistic poses according to Table 8.8. It is even more detrimental to the modeling of global positions. Consequently, we will keep dropout disabled.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
1 layer	71.4	110.3	80.4	67.4	0.557	0.442
2 layers	36.1	75.0	91.9	78.2	0.736	0.568
3 layers	**34.8**	**74.3**	**92.3**	**78.5**	**0.744**	**0.573**

Table 8.6.: Relative error metrics for GRU with different number of layers.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
$p = 0.0$	**41.7**	**104.8**	**90.5**	**67.0**	**0.694**	**0.451**
$p = 0.01$	43.0	107.2	90.1	66.4	0.688	0.446
$p = 0.1$	45.5	111.2	89.3	65.5	0.676	0.437

Table 8.7.: Absolute error metrics for GRU with different dropout probabilities p.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
$p = 0.0$	**36.1**	**75.0**	**91.9**	**78.2**	**0.736**	**0.568**
$p = 0.01$	36.2	75.3	**91.9**	78.1	**0.736**	**0.568**
$p = 0.1$	36.9	76.0	91.6	77.9	0.732	0.566

Table 8.8.: Relative error metrics for GRU with different dropout probabilities p.

8.2.3. Dimension of Hidden State

From Table 8.9, we can see that Martinez et al. choice of 1024 latent features is indeed appropriate. Less cell state memory clearly hurts the performance. Interestingly enough, more cell memory seems to pose a problem as well, possibly due to the lack of regularization which was not needed prior to increasing the hidden state dimensionality. The relative errors in Table 8.10 do not provide any additional insights.

8.2.4. Pose Representation

The different input representations (see Section 6.2) have vastly different performance results. Until now, we have only used bone vectors. From Table 8.11 and Table 8.12, we can see that bone vectors and a mixed representation obtain somewhat similar results. Bone vectors are better both in the absolute and relative sense.
Unsurprisingly, modeling each joint in world coordinates does not work well. The

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
$h = 512$	48.4	115.2	88.0	63.8	0.658	0.422
$h = 1024$	**41.7**	**104.8**	**90.5**	**67.0**	**0.694**	**0.451**
$h = 2048$	1378.5	1390.5	0.1	0.1	0.000	0.000

Table 8.9.: Absolute error metrics for GRU with different hidden state dimensions h.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
$h = 512$	40.2	79.7	90.4	76.5	0.714	0.551
$h = 1024$	**36.1**	**75.0**	**91.9**	**78.2**	**0.736**	**0.568**
$h = 2048$	1262.1	1275.6	4.3	4.3	0.042	0.042

Table 8.10.: Relative error metrics for GRU with different hidden state dimensions h.

problem is that each joint must be moved correctly through space but also positioned correctly with respect to its neighbors, whereas for bone vectors or mixed poses, each non-root joint must only be positioned correctly with respect to a parent joint. The difficult problem of moving all points as a connected unit is avoided by only moving the root joint and applying that movement to all other joints implicitly.
It is rather unclear why just modeling relative joints works poorly but apparently the parameter setting favors some sort of absolute movement as well.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
absolute	85.3	166.8	74.0	49.8	0.501	0.310
bones	**41.7**	**104.8**	**90.5**	**67.0**	**0.694**	**0.451**
mixed	44.9	108.5	89.3	65.9	0.677	0.440
relative	-	-	-	-	-	-

Table 8.11.: Absolute error metrics for GRU with different input representations.

8.2.5. Batch Size

For the GRU architecture, Table 8.13 and Table 8.14 clearly show that large batch sizes work best. Consequently, we stick to the batch size of 128. Like in some of the previous analyses, we can see that the GRU architecture is rather

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
absolute	69.0	111.2	79.8	66.7	0.596	0.464
bones	**36.1**	**75.0**	**91.9**	**78.2**	**0.736**	**0.568**
mixed	38.4	77.1	91.1	77.6	0.724	0.561
relative	73.5	115.6	78.6	65.7	0.575	0.447

Table 8.12.: Relative error metrics for GRU with different input representations.

unstable with respect to certain design choices, i.e., performance varies by a large margin when changing certain parameters.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
$b = 64$	93.0	179.7	70.7	47.1	0.468	0.287
$b = 128$	**41.7**	**104.8**	**90.5**	**67.0**	**0.694**	**0.451**
$b = 256$	45.2	110.0	89.2	65.5	0.675	0.437

Table 8.13.: Absolute error metrics for GRU with different batch sizes b.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
$b = 64$	70.1	112.3	79.5	66.4	0.592	0.461
$b = 128$	**36.1**	**75.0**	**91.9**	**78.2**	**0.736**	**0.568**
$b = 256$	37.9	77.2	91.3	77.4	0.726	0.560

Table 8.14.: Relative error metrics for GRU with different batch sizes b.

8.2.6. Loss Function

According to Table 8.15, the L2 loss penalizes difficult joints too much so that the network seems to learn a a low-risk prediction, which leads to a lower accuracy compared to a more balanced L1 loss term. The result is similarly present in relative predictions. Therefore, we will keep using the L1 loss.

8.2.7. Data Normalization

We mentioned in Section 6.4.1 that data normalization mainly consists of a centering translation of the point cloud of all joints at all frames and an additional feature-wise normalization. The centering translation is always performed but

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
L1 loss	**41.7**	**104.8**	**90.5**	**67.0**	**0.694**	**0.451**
L2 loss	48.2	112.6	89.1	63.7	0.637	0.398

Table 8.15.: Absolute error metrics for GRU with different loss functions.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
L1 loss	**36.1**	**75.0**	**91.9**	**78.2**	**0.736**	**0.568**
L2 loss	41.7	80.8	90.7	76.0	0.687	0.524

Table 8.16.: Relative error metrics for GRU with different loss functions.

we want to investigate the importance of zero-mean unit-variance features. Table 8.17 and Table 8.18 show a massive increase in performance across all metrics so that we conclude that centered features are crucial for good results. Martinez et al. [MBR17] employed feature normalization as well.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
w/ normalize	**41.7**	**104.8**	**90.5**	**67.0**	**0.694**	**0.451**
w/o normalize	60.8	131.3	84.3	58.5	0.592	0.368

Table 8.17.: Absolute error metrics for GRU with and without normalization.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
w/ normalize	**36.1**	**75.0**	**91.9**	**78.2**	**0.736**	**0.568**
w/o normalize	52.3	92.9	86.9	72.4	0.650	0.503

Table 8.18.: Relative error metrics for GRU with and without normalization.

8.2.8. Summary of RNN Tuning

We will shortly summarize the most important discoveries of how our RNN architectures behaved. We saw that LSTMs are significant worse than GRUs on the validation data. Furthermore, we determined that GRUs are quite sensitive to many settings, especially the number of parameters. The one-layered GRU was

outperformed by its corresponding LSTM, and too large cell states are detrimental as well. However, using regularization on GRUs with the right size should be avoided as well.

8.3. Transformer Tuning

The next hyperparameter study will focus on finding good settings for training transformers to perform motion prediction. The study is as thorough as Section 8.2 but we again refer to Appendix A for redundant or less interesting results.

8.3.1. Number of Layers

From Table 8.19 and Table 8.20, we can conclude that four layers are not enough for optimal performance. Likewise, using ten layers yields not advantage justifying the additional complexity. The decision between six and eight layers is rather ambiguous. The performance increase with the two additional layers could result from experimental noise or a stronger model. When confronted with a similar decision for the number of stacked GRU layers (see Section 8.2.1), we decided in favor of the smaller model because of the 50 % increase of training duration. However, the eight-layered transformers achieved a much higher GPU utilization than those with four or six layers so that the additional complexity does not matter that much. In fact, many eight-layered networks trained faster (about 90 min) than the six-layered one (105 min) although this should be taken with a grain of salt because execution times on Claix18 naturally varied. Taking this into account, we decided for the eight-layered transformer.
Training results are not reproducible with newer code versions, because the initial order of the training data depended on the hardware.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
4 layers	53.5	122.2	85.7	61.7	0.636	0.410
6 layers	51.1	118.4	86.7	62.9	0.647	0.420
8 layers	**50.6**	**117.5**	**86.8**	**63.2**	**0.649**	**0.422**
10 layers	51.0	118.1	**86.8**	63.1	0.648	0.421

Table 8.19.: Absolute error metrics for transformer with different number of layers.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
4 layers	43.5	85.3	89.0	74.6	0.702	0.539
6 layers	41.9	83.1	**89.6**	75.3	0.709	0.546
8 layers	**41.8**	**82.7**	**89.6**	**75.4**	0.709	0.546
10 layers	42.0	82.9	**89.6**	**75.4**	**0.710**	**0.547**

Table 8.20.: Relative error metrics for transformer with different number of layers.

8.3.2. Dropout

Since the transformer architecture is much deeper than the GRU, we analyze higher dropout probabilities than for GRUs. Similar to the results in Section 8.2.2, high dropout probabilities seem to be problematic both for relative pose quality and for global motion, as can be seen in Table 8.21 and Table 8.22. Indeed, our initial choice of 10 % turned out to be too high.
Training results are not reproducible with newer code versions, because the initial order of the training data depended on the hardware.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
$p = 0.05$	**49.0**	**115.1**	**87.5**	**63.9**	**0.658**	**0.428**
$p = 0.1$	50.6	117.5	86.8	63.2	0.649	0.422
$p = 0.2$	54.8	124.3	85.3	61.5	0.630	0.407

Table 8.21.: Absolute error metrics for transformer with different dropout probabilities p.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
$p = 0.05$	**40.6**	**81.4**	**90.1**	**75.9**	**0.717**	**0.552**
$p = 0.1$	41.8	82.7	89.6	75.4	0.709	0.546
$p = 0.2$	44.4	86.2	88.6	74.3	0.696	0.536

Table 8.22.: Relative error metrics for transformer with different dropout probabilities p.

8.3.3. Weight Decay

In Table 8.23 and Table 8.24, we can see that both types of error are improved when using a weight decay coefficient of 0.01. The improvement is only marginal,

but large enough to exceed normal statistical variation. In the previous experiments, weight decay was disabled which may therefore be not optimal.
Training results are not reproducible with newer code versions, because the initial order of the training data depended on the hardware. This experiment still uses a dropout probability of 10 %.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
$w = 0$	50.6	117.5	86.8	63.2	0.649	0.422
$w = 10^{-3}$	50.9	117.8	86.7	63.1	0.647	0.421
$w = 10^{-2}$	**49.9**	**116.3**	**87.1**	**63.5**	**0.653**	**0.425**

Table 8.23.: Absolute error metrics for transformer with different weight decay coefficients w.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
$w = 0$	41.8	82.7	89.6	75.4	0.709	0.546
$w = 10^{-3}$	42.0	83.2	89.5	75.4	0.709	0.546
$w = 10^{-2}$	**41.1**	**82.0**	**89.9**	**75.7**	**0.713**	**0.550**

Table 8.24.: Relative error metrics for transformer with different weight decay coefficients w.

8.3.4. Dimension of Embedding Space

Changing the internal feature space dimension leads to similar behavior as in Section 8.2.3 for GRUs. A smaller feature space leads to slightly worse absolute and relative errors in Table 8.25 and Table 8.26, while a larger feature space converges to massively worse solutions. However, the variations are, not for the last time, less extreme than for GRUs, where the larger cell state did not converge at all.
These and all following results on transformers are fully reproducible. From now on, we are using an eight-layered transformer with lowered dropout of 5 % and a weight decay factor of 0.01, i.e., we chose the best setting for each of the three previously analyzed hyperparameters. Indeed, applying higher weight decay and lower dropout at the same time improves the overall performance compared to only applying one of these improvements (observe that the MPJPE for $h = 128$ in Table 8.25 decreased compared to all models in Table 8.21 and in Table 8.23). Note that we did not experiment with even lower dropout rates or higher weight decays, but instead refer to Section 10.2.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
$h = 64$	50.6	116.8	86.9	63.3	0.651	0.423
$h = 128$	**47.6**	**112.9**	**88.0**	**64.3**	**0.664**	**0.432**
$h = 256$	136.7	282.5	58.1	40.5	0.404	0.273

Table 8.25.: Absolute error metrics for transformer with different embedding space dimensions h.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
$h = 64$	42.2	83.1	89.5	75.2	0.709	0.546
$h = 128$	**39.8**	**80.3**	**90.4**	**76.2**	**0.720**	**0.554**
$h = 256$	77.0	125.3	77.3	64.1	0.581	0.455

Table 8.26.: Relative error metrics for transformer with different embedding space dimensions h.

8.3.5. Pose Representation

The underlying pose representation is a parameter independent from the choice between transformer and GRU so one would expect similar results for both architectures. Indeed, bone vectors again turn out to be extremely powerful so that we will keep using this representation, see Table 8.27. The mixed poses seems to be slightly better, but the improvement is only marginal and may therefore not be of statistical significance.
An important difference to the GRUs is that the choice of input representation is significantly more crucial to the GRU than to the transformer, i.e., Table 8.28 shows that transformers with absolute joint locations and root-relative joint locations do not perform significantly worse in terms of relative pose quality. The relative representation even obtains the highest quality in relative pose configuration which is something we also expected to see for the GRU. However, relative poses are obviously inferior when considering global movement so that we will not consider them in future experiments.

8.3.6. Learning Rate Warm-Up

Many authors [ACKH20, VSP+17] report that an alternative learning rate schedule with an initial warm-up phase of increasing learning rates followed by an exponential decay is beneficial to the performance. We adopted the algorithm from [VSP+17] but found it to be harmful for modeling global motion (see Ta-

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
absolute	48.0	113.8	87.9	64.0	0.664	0.430
bones	47.6	112.9	88.0	64.3	0.664	0.432
mixed	**47.1**	**112.3**	**88.2**	**64.5**	**0.667**	**0.433**
relative	-	-	-	-	-	-

Table 8.27.: Absolute error metrics for transformer with different input representations.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
absolute	40.6	81.7	90.1	75.7	0.716	0.549
bones	39.8	80.3	90.4	76.2	0.720	0.554
mixed	39.4	79.9	90.5	76.4	0.722	0.556
relative	**38.9**	**79.4**	**90.6**	**76.5**	**0.725**	**0.558**

Table 8.28.: Relative error metrics for transformer with different input representations.

ble 8.29). We therefore kept our original learning rate starting at 0.001 and using an exponential decay with factor 0.98.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
w/ warm-up	50.2	117.4	87.0	63.1	0.650	0.419
w/o warm-up	**47.6**	**112.9**	**88.0**	**64.3**	**0.664**	**0.432**

Table 8.29.: Absolute error metrics for transformer with and without warming up the learning rate.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
w/ warm-up	41.3	81.9	89.8	75.6	0.713	0.548
w/o warm-up	**39.8**	**80.3**	**90.4**	**76.2**	**0.720**	**0.554**

Table 8.30.: Relative error metrics for transformer with and without warming up the learning rate.

8.3.7. Summary of Transformer Tuning

In Section 8.2.8, we have seen that GRUs have much potential, but are in turn quite susceptible to certain settings like the pose representation. In contrast, the transformer has almost the same performance with all different input types. Like the GRUS, a latent space with too many dimensions is detrimental, but to a much lesser degree than for GRUs. Lastly, we saw that regularization is beneficial to transformers but not to GRUs.

8.4. Comparison

With our tuned architectures, we can now move on to evaluating the performance on independent test data. This section will contain quantitative comparisons between the methods and also consider a state of the art method presented by Mao et al. [MLS20], which uses the same evaluation protocol. We will investigate the same performance metrics as before, but unlike before, we will remove global rotation in the relative setting for better comparability to [MLS20].

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
Zero Motion	149.8	254.7	58.2	**53.4**	0.417	**0.377**
Const. Velocity	152.4	444.9	59.1	27.0	0.378	0.166
LSTM	158.2	366.7	55.5	36.8	0.401	0.240
GRU	**70.8**	**152.2**	**78.7**	46.9	**0.519**	0.274
Transformer	118.4	200.9	54.6	37.2	0.349	0.237

Table 8.31.: Absolute error metrics for the different methods. The metrics show the performance at the corresponding point in time, i.e., no temporal averaging is used.

Table 8.31 shows the performances for motion prediction with global orientation and movement. Since [MLS20] removes global information, it is not represented in the table.
The first thing to notice is that the performance between short term and long term prediction can differ a lot across methods, e.g., when looking at the percentage of correct keypoints. Especially the constant velocity baseline (see Section 5.3) is quite competitive for short term goals but has a horrible performance in the long run. This is not surprising since there are many joint constraints which limit how long a motion can continue. In contrast, the zero-motion baseline from [MBR17] almost loses no accuracy over the course of 600 ms. This demonstrates that the majority of joints does not move more than 10 cm during the predicted time horizon. The RNN and transformer architectures are somewhat in the middle. Both recurrent networks lose a little bit more accuracy over time and the transformer

starts with a lower PCK but is able to retain it better. Overall, the PCK suggests to either use the GRU model or the zero motion baseline heuristic. For short term predictions, the constant velocity baseline works as well.

When looking at PCK AUC, the results are extremely similar. Zero motion baseline and GRU are still by far the best methods. Some nuances differ, e.g., the constant velocity baseline looks worse under PCK AUC and the LSTM seems to work a bit better compared to PCK. As explained in Section 7.1.2, the PCK AUC is less of a binary decision and more of a continuous MPJPE ignoring outliers. Note that the PCK AUC threshold and the PCK threshold are close together so that this difference is not explained by the outliers. Thus, the LSTM might not be correct as often (lower PCK than constant velocity baseline), but when it is, it is consistently more accurate (higher PCK AUC compared to baseline). The transformer architecture is rather bad under both metrics.

This changes drastically when looking at MPJPE. Here, the GRU is the best method by a large margin. The transformer comes second; it is definitely less accurate than the GRU but still significantly better than the rest. The LSTM and the two baselines have poor performance, especially for long-term prediction. Hence, we conclude that LSTM and baselines occasionally fail completely which then results in MPJPE scores that do not reflect the average performance. In summary, the GRU is the best method. The transformer is comparable to other approaches under PCK but has really good performance on MPJPE.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
Zero Motion	107.3	114.8	67.2	64.7	0.437	0.422
Const. Velocity	148.1	392.8	55.5	22.4	0.326	0.121
LSTM	43.7	73.1	**91.0**	**78.9**	**0.669**	**0.530**
GRU	54.9	91.3	85.2	71.8	0.593	0.466
Transformer	69.8	92.5	78.0	69.5	0.529	0.446
[MLS20]	**42.0**	**67.2**	-	-	-	-

Table 8.32.: Relative error metrics for the different methods. Global rotation is removed using all joints. The metrics show the performance at the corresponding point in time, i.e., no temporal averaging is used.

When comparing to the literature, one difficulty is that existing methods remove global rotation before applying forward kinematics. Since we trained our models on data with global movement, we therefore need to align ground truth and predicted pose, which is explained in Section 7.2. We therefore provide a

relative pose quality upper bound in Table 8.32 and a lower bound in Table 8.33. The results within each table are extremely similar and the only difference is how close the performance is to the state of the art method. We therefore focus on Table 8.32. One thing to immediately notice is that the performances are maximally consistent across different time horizons and different metrics. It is therefore extremely easy to rank our methods from best to worst: our best method is the LSTM, followed by GRU and transformer, whereas both baselines have extremely poor relative quality. We emphasize that the difference between transformer and GRU are minuscule for long-term prediction.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
Zero Motion	111.4	119.4	66.2	64.0	0.460	0.439
Const. Velocity	158.0	425.7	57.2	27.8	0.355	0.150
LSTM	49.7	79.2	**88.6**	**76.8**	**0.637**	**0.513**
GRU	61.0	97.3	81.9	70.1	0.573	0.455
Transformer	76.4	100.0	75.3	67.0	0.517	0.437
[MLS20]	**42.0**	**67.2**	-	-	-	-

Table 8.33.: Relative error metrics for the different methods. Global rotation is removed using the minimum number of three joints. The metrics show the performance at the corresponding point in time, i.e., no temporal averaging is used.

There are two things to notice here. Firstly, these results seem paradox compared to our hyperparameter study. On the validation split, the LSTM was outperformed by the GRU in terms of relative pose quality by a significant amount of 5 mm. Furthermore, it was at par with the transformer (also in terms of relative pose quality). Note that we specifically only applied improvements to GRU and transformer if the performance increased significantly, e.g., at least 1 mm decrease of absolute MPJPE. Therefore, the extreme performance gap of 18 mm on test data is hard to explain. The best solution would be to gather more unseen data in order to find out whether validation or testing split are more representative of the real generalization performance.

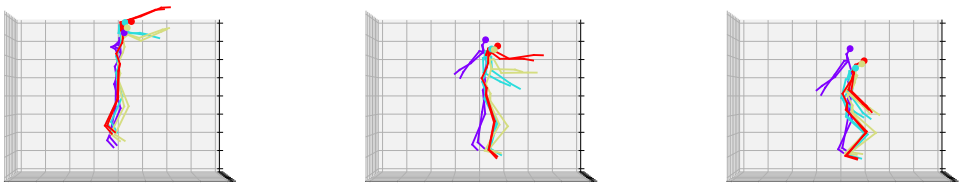
The second thing to investigate is the difference between relative and absolute error. Our GRU outperforms the LSTM across all absolute metrics but is outperformed across all relative metrics. This strongly indicates that the tuned LSTM is worse at modeling global motion but shines at relative pose quality. However, it also contrasts the fact that Martinez et al. preferred a GRU design over an LSTM. One possibility is that the experimental set-up in [MBR17] is more simi-

lar to our hyperparameter study than to the comparison currently presented.
Note that one reason for the bad performance of GRU and transformer might be that the test split defined by Mao et al. contained many treadmill exercises. If those were underrepresented in the training split, the learned dependency between moving legs and changes in global positioning would generalize badly to the test data. We unfortunately do not know whether training data contained treadmill motions.
From Table 8.33 and Table 8.32, we can see that our LSTM inspired by [MBR17] is close to state of the art for short term prediction since it is between 1.7 mm and 7.7 mm worse. For long term prediction, the gap lies between 5.9 mm and 12 mm which is still considerable but not insurmountable. It is important to keep in mind that our models were trained to be aware of global orientation and position so that Mao's method has a critical advantage.

8.5. Qualitative Evaluation

Although qualitative evaluation may not be representative of the real performance at all, it is nevertheless important to ensure that the deep learning framework did not find loopholes in the task formulation. We will therefore provide two examples of visual results.

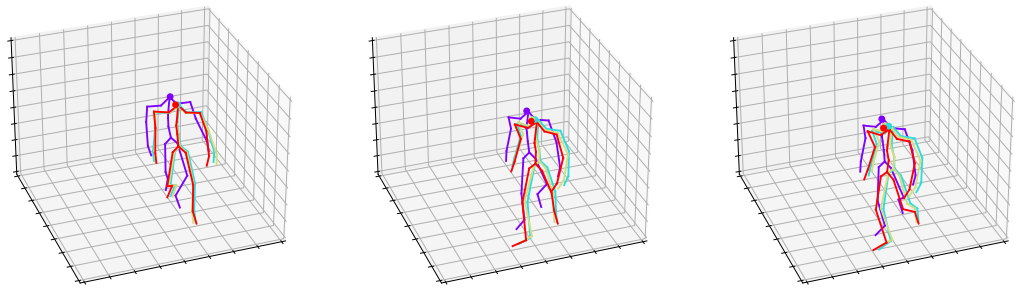
In Figure 8.1, we can see a human at different points in time during a jump. The apex of the jump is reached at about 0.4 s. All three prediction methods correctly model that the legs fully extend in the air. Furthermore, all methods predict arm positions which are realistic during a jump, although those from the transformer (purple) do not correspond to what the subject is actually doing. We can furthermore see that the transformer either has a temporal delay or gets stuck in the animation. Overall, the LSTM (cyan) predicts the height during the apex best (compare to ground truth in red).



(a) Jumping after 0.4 s. (b) Jumping after 0.8 s. (c) Jumping after 1.0 s.

Figure 8.1.: Human jumping. Ground truth is red, transformer purple, GRU beige and LSTM cyan.

Walking is a periodic action where convergence against a still pose is not desired. Figure 8.2 demonstrates that none of the three methods does that. Moreover, we can see that all three models predict steps synchronized with the ground truth (red). Both RNNs are highly accurate, whereas the transformer predicts positions which are slightly off. The residual seems to be constant over time which indicates that the transformer still learns realistic trajectories.



(a) Walking after 0.4 s. (b) Walking after 0.8 s. (c) Walking after 1.0 s.

Figure 8.2.: Human walking. Ground truth is red, transformer purple, GRU beige and LSTM cyan.

8.6. Final Summary

We have seen in Section 8.2.8 and Section 8.3.7 that GRUs perform well on validation data but are harder to tune than transformers. In the quantitative comparison, it turned out that LSTMs can be quite powerful as well. We emphasize that those models (GRU and transformer) with good global positioning are much farther away from state of the art than the LSTM with mediocre positioning. This highlights the difficulty of global components in motion forecasting and demands for further research. In summary, our best method is the adopted GRU, as it performs well over a wide variety of metrics. Under some metrics, our proposed transformer can achieve comparable results to the GRU. All three models created realistic poses in our visualizations.

9 Results on Multi-Person Data

One of the main focuses of the master thesis is how absolute positioning in space can be improved from considering interacting humans jointly. This chapter is dedicated to analyzing these effects quantitatively. The hyperparameters are mostly taken from Chapter 8, but we revisit a few of them for the multi-person data from PKU-MMD and NTU-RGB+D 120. Most of them are analyzed in Appendix B. We will also provide an ablation study on our person attention module. Lastly, we will compare our different methods on the test data quantitatively and qualitatively.

9.1. Transformer Tuning

Since a complete tuning was already conducted for AMASS, we focus on those hyperparameters related to the Kinect datasets. Here, we will present relevant tables for transformers. The results for GRUs as well as a further comparison for transformer do not provide new insights, therefore we refer to Appendix B.

9.1.1. Dimension of Embedding Space

The datasets NTU-RGB+D 120 and PKU-MMD are significantly smaller than AMASS and we previously saw, e.g., in Section 8.3.4, that too many weights can be devastating to the performance. Therefore, Table 9.1 and Table 9.2 demonstrate that an embedding space dimension of 64, which was noticeably too small for AMASS, is almost competitive with the default size of 128. However, it also does not improve the performance so that we will keep the default value.

9.1.2. Ablation of Person Attention

Perhaps the most important comparison is presented in Table 9.3, which shows the performance gain of modeling both interacting humans jointly. The usage

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
$h = 64$	38.5	55.8	91.2	84.0	0.700	0.614
$h = 128$	**37.5**	**54.4**	**91.6**	**84.5**	**0.704**	**0.619**

Table 9.1.: Absolute error metrics for transformer with different embedding space dimensions h.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
$h = 64$	27.3	39.7	94.1	89.5	0.789	0.718
$h = 128$	**26.6**	**38.5**	**94.4**	**90.0**	**0.792**	**0.723**

Table 9.2.: Relative error metrics for transformer with different embedding space dimensions h.

of person attention leads to small improvements for short term predictions but the long term predictions is what profits most. The percentage of correct joints increases by 2 % and the MPJPE decreases by 6 mm. This might not seem like much compared to some of the variations on AMASS, but the Kinect datasets are considerably easier so that improvement is likely more difficult to achieve. In fact, improvements of 10 % like this one only occurred for transformers when a hyperparameter was chosen significantly too high, e.g., 20 % dropout probability instead of 5 %. Most importantly, the best GRU, which is provided for comparability, outperforms the single-person transformer but cannot keep up with the person attention under MPJPE. We therefore conclude that the performance advantage of the GRU in comparison to the transformer is undone by multi-person modeling which was the goal of our work.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
w/ attention	**37.5**	**54.4**	**91.6**	**84.5**	**0.704**	**0.619**
w/o attention	39.6	60.5	90.6	82.8	0.696	0.608
best GRU	38.4	57.3	91.4	84.2	0.700	0.615

Table 9.3.: Absolute error metrics for transformer with and without person attention. Best GRU performance is shown for comparability.

We want to mention that relative pose quality does not benefit nearly as much from person attention, see Table 9.4. As mentioned before, important constraints

on absolute positioning can be derived from interactions, e.g., during a handshake, the right palms have to touch, which restricts the distance between the two individuals. However, the rest of the pose cannot be directly inferred from a handshake. Consequently, it should not be surprising that the relative pose accuracy of transformer and GRU are within statistical noise.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
w/ attention	26.6	38.5	94.4	90.0	**0.792**	0.723
w/o attention	27.1	39.7	94.2	89.5	0.790	0.719
best GRU	**26.4**	**38.1**	**94.6**	**90.4**	0.791	**0.724**

Table 9.4.: Relative error metrics for transformer with and without person attention. Best GRU performance is shown for comparability.

9.2. Comparison

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
Zero Motion	44.7	76.0	88.3	79.8	0.694	0.599
Const. Velocity	84.6	195.0	76.7	52.2	0.520	0.314
LSTM	34.0	53.8	92.9	85.1	0.729	0.633
GRU	33.4	52.5	**93.2**	85.5	0.732	0.636
Transformer	**33.1**	**50.9**	**93.2**	**85.7**	**0.735**	**0.641**

Table 9.5.: Absolute error metrics for the different methods.

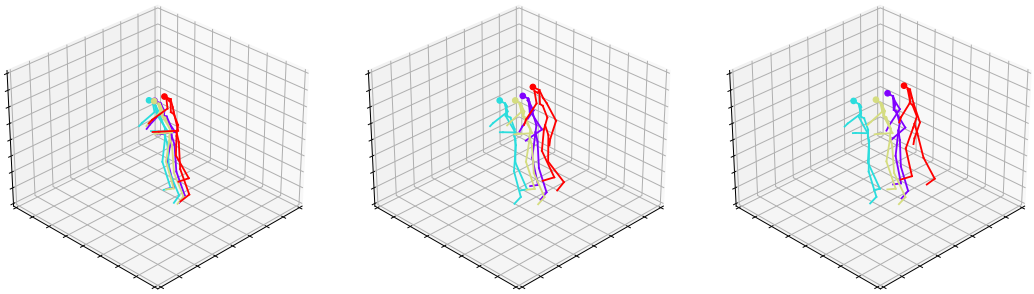
In our final quantitative evaluation, we compare our three models and the two baselines (see Section 8.4) on the Kinect test data. Both RNNs and the transformer outperform the baselines absolutely in Table 9.5 and relatively in Table 9.6. The constant velocity baseline is not even competitive for short term predictions. The zero-motion baseline performs acceptable for relative PCK but is still outperformed by 2 %.
Table 9.6 shows that all three networks achieve the same quality for relative poses, which is at stark contrast to Section 8.4 where the LSTM was significantly better for relative accuracy. This indicates that LSTMs are probably not intrinsically better at relative performance. Instead, that specific setup of LSTM training may have resulted in a particularly accurate model for relative motion estimation or in a model that generalizes exceptionally well to the data distribution of the test set, or a combination of both.

Lastly, we can see in Table 9.5 that the transformer with person attention slightly outperforms both RNNs. The performance gain is more apparent for long term prediction and becomes most evident under MPJPE. Keep in mind that on AMASS, transformers performed significantly worse than the GRUs. This boost can be explained by the person attention module, which is also reinforced by the fact that the transformer without person attention in Table 9.3 performed worse than the best GRU on the validation data. We therefore conclude that transformers with person attention perform best on the Kinect datasets.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
Zero Motion	27.8	44.1	93.7	88.3	0.801	0.725
Const. Velocity	49.9	115.7	88.4	71.9	0.691	0.505
LSTM	22.7	36.3	95.7	90.8	0.818	0.740
GRU	**22.4**	**35.6**	**95.9**	**91.1**	0.819	0.742
Transformer	22.5	**35.6**	95.7	90.8	**0.820**	**0.744**

Table 9.6.: Relative error metrics for the different methods.

9.3. Qualitative Evaluation

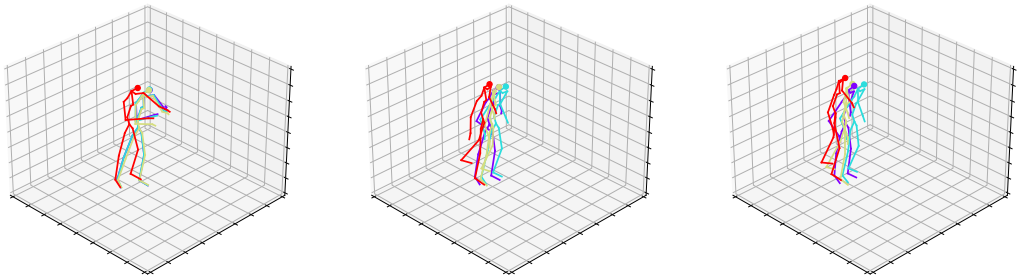


(a) Hugging after 0.4 s. (b) Hugging after 0.8 s. (c) Hugging after 1.0 s.

Figure 9.1.: Two people after hugging. The second person is not visualized. Ground truth is red, transformer purple, GRU beige and LSTM cyan.

Again, we conclude the comparison with visual results. Figure 9.1 shows one of two individuals hugging each other. The prediction interval contains a part of the hugging and the process of releasing the other person, although the former is not apparent from the images. We can clearly see that the transformer (purple) predicts the most accurate global positions as it is significantly closer to the ground truth (red). The LSTM (cyan) performs worst. All three methods predict

similar relative pose configurations, where each one is plausible.
The second person of the interaction is visualized in Figure 9.2. Here, we can see that all three methods correctly predict how the person lowers the arms. In contrast, only the GRU correctly predicts the step backwards but its global location prediction is only slightly superior.



(a) Hugging after 0.4 s. (b) Hugging after 0.8 s. (c) Hugging after 1.0 s.

Figure 9.2.: Two people after hugging. This visualization shows the other person. Ground truth is red, transformer purple, GRU beige and LSTM cyan.

9.4. Summary

Compared to Section 8.4, the three architectures lead to much closer performances on the Kinect datasets. The most likely reason is that the Kinect data is easier than AMASS, which can be seen in the significantly higher accuracy of each method. Moreover, the performance between LSTM and GRU is now within a range that we expected. Further tests are nevertheless required to understand the performance difference on AMASS.
Again, the accuracy of the transformer was low compared to the RNNs. With the help of our new person attention module, we were able to completely close the gap. We can therefore conclude that multi-person motion forecasting should indeed be investigated more closely, e.g., one might extend GRUs with person attention, see also Section 10.2.

10 Conclusion

In the previous chapters, we have extensively shown how model size, regularization, etc., affect our attention-based method and the adopted approach using recurrent networks. Moreover, we have compared our methods with baseline heuristics and literature, which demonstrated that the methods converge against largely different solutions. Lastly, we will summarize our most important results and formulate future questions that came up during our research.

10.1. Summary

Our main research question was how motion forecasting profits from modeling multiple people jointly. We have seen that our transformer architecture, which failed to compete on the single-person dataset AMASS, was able to perform on par with the recurrent architectures on the enhanced multi-person dataset. The ablation showed that the performance gain was most prominent in the absolute sense, i.e., when modeling global motion, which matches our expectation because it allows to spatially group interacting humans. Therefore, we conclude that motion forecasting definitely benefits from multi-person modeling.
We have also seen that it is hard to determine a best model. Our LSTM variant under-performed on the AMASS validation split and in an absolute sense, but came closest to state of the art on the AMASS test split when global movement was removed. On the other hand, the GRU excelled in modeling global motion across datasets. We emphasize that this research question cannot be fully answered in the scope of this work, e.g., Martinez et al. preferred a GRU for relative modeling, which seems to contradict our results. Nevertheless, the GRU architecture performed well in a wide variety of settings, so that we want to highlight its versatility and recommend it as a good starting point, although we found it to be quite unstable with respect to many hyperparameters like the model size or the underlying pose representation.
Additionally, we want to reinforce our claim that frameworks should not discard

global motion during forecasting. We have seen that one of our models performs close to state of the art despite being trained on a more difficult task. Furthermore, we observed great disparities in absolute and relative performance between our models so that we conclude that motion prediction with global movement components is far from trivial. Nevertheless, we argue that localization of humans and motion forecasting are inherently linked tasks.

10.2. Future Work

Lastly, let us discuss future directions for research based on the results we presented. Our suggestions can be summarized into three groups.
Firstly, we recommend to further analyze the result that was most surprising to us. If GRUs are both absolutely and relatively better than LSTMs on our validation split, then why does the LSTM perform closest to state of the art? These extreme variations in performance seem suspicious to us. An easy method to investigate this question is to design multiple new training-validation splits and check for each of these splits how LSTMs and GRUs perform. Additionally, one should use different random seeds and average the performances.
Secondly, we suggest to investigate the accuracy of the transformer more closely. We note that we spend significantly more time tuning the RNNs compared to the transformer due to time constraints so that our results in that regard are not yet set in stone. Some starting points based on our results are weight decay and dropout, e.g., one could use different dropout probabilities for attention scores and for other layers. Pre-training on an auxiliary task might improve results as well. Most importantly, we recommend to investigate how our transformer performs with sampling-based auto-regressive generation analogue to the GRU. We argued that our formulation is faster but were not able to compare the accuracy as that would require a custom attention implementation in order to cache old queries, keys and values.
Finally, we observed that absolute and relative error are not always correlated. This leads to the question whether these tasks should be separated more clearly. A draft design might contain a recurrent network for relative forecasting and a smaller transformer network with person attention for tracking global positions. These two networks could then exchange intermediate states since we still argue that both tasks inherently depend on each other.

A

Further Results on AMASS Dataset

This chapter is dedicated to provide tables on those hyperparameters with little influence on performance or those which are redundant between the GRU tuning and the transformer tuning.

A.1. GRU

First, let us consider the remaining hyperparameters in the GRU experiments.

A.1.1. Weight Decay

From Table A.1 and Table A.2, it becomes evident that increasing amounts of weight decay do not really affect accuracy of the predictions. This is backed up by the fact that no dropout is used, meaning that regularization does not seem necessary for GRUs in motion forecasting.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
$w = 0$	41.7	104.8	**90.5**	67.0	0.694	0.451
$w = 10^{-3}$	**41.6**	**104.7**	**90.5**	67.0	0.694	0.450
$w = 10^{-2}$	41.7	105.1	**90.5**	**67.1**	**0.695**	**0.452**

Table A.1.: Absolute error metrics for GRU with different weight decay coefficients w.

A.1.2. Absolute and Relative Loss

The balancing between absolute and relative pose loss, see Section 4.3.2, determines how important global tracking and accurate relative poses are compared

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
$w = 0$	36.1	**75.0**	**91.9**	**78.2**	**0.736**	0.568
$w = 10^{-3}$	**36.0**	**75.0**	**91.9**	**78.2**	**0.736**	0.568
$w = 10^{-2}$	36.1	75.2	**91.9**	**78.2**	**0.736**	**0.569**

Table A.2.: Relative error metrics for GRU with different weight decay coefficients w.

to each other. Unsurprisingly, relative poses are not more accurately when the absolute loss term is weighted highly (Table A.4). Vice versa, Table A.3 shows that low values for α lead to bad tracking quality. A perfectly balanced value between the two terms provides a considerable increase in modeling global motion without hurting relative accuracy by too much. We therefore keep the loss terms balanced.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
$\alpha = 0.1$	46.4	113.8	89.0	64.6	0.668	0.428
$\alpha = 0.5$	**41.7**	104.8	**90.5**	67.0	**0.694**	**0.451**
$\alpha = 0.9$	**41.7**	**104.0**	**90.5**	**67.1**	**0.694**	**0.451**

Table A.3.: Absolute error metrics for GRU with different importance α of the absolute loss term.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
$\alpha = 0.1$	**35.9**	**75.0**	**91.9**	**78.2**	**0.737**	**0.570**
$\alpha = 0.5$	36.1	**75.0**	**91.9**	**78.2**	0.736	0.568
$\alpha = 0.9$	37.3	76.5	91.6	77.8	0.727	0.560

Table A.4.: Relative error metrics for GRU with different importance α of the absolute loss term.

A.1.3. Weight Initialization

Glorot et al. [GB10] derived that network weights should be initialized to preserve variance. Since the default PyTorch deviates from the recommendation, we decided to compare the results. From Table A.5 and Table A.6, we can see that

the PyTorch variant works better for our use case so that we keep the default initialization active.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
w/ Glorot	122.9	228.7	58.6	35.9	0.326	0.186
w/o Glorot	**41.7**	**104.8**	**90.5**	**67.0**	**0.694**	**0.451**

Table A.5.: Absolute error metrics for GRU with and without Glorot weight initialization.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
w/ Glorot	82.9	121.8	75.5	62.6	0.501	0.393
w/o Glorot	**36.1**	**75.0**	**91.9**	**78.2**	**0.736**	**0.568**

Table A.6.: Relative error metrics for GRU with and without Glorot weight initialization.

A.1.4. Learning Rate

The learning rate is generally considered to be extremely important. Table A.7 highlights that the default learning rate of Adam [KB15] works best in our case. Note that Adam internally rescales the gradients anyways, which may be a reason why 0.001 seems to be a good choice when using Adam. All three networks utilize the same exponential learning rate schedule with an additional decay by factor ten shortly before the end of training.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
$l = 2 \times 10^{-3}$	47.4	113.8	88.4	64.2	0.662	0.425
$l = 5 \times 10^{-4}$	**41.7**	**104.8**	**90.5**	**67.0**	**0.694**	**0.451**
$l = 1 \times 10^{-3}$	97.8	182.1	67.9	46.2	0.444	0.283

Table A.7.: Absolute error metrics for GRU with different learning rates l.

A.1.5. Length of Input

Methods like [MBR17, MLS20] often feed 2 s of conditioning input to the network but never motivate why. Table A.9 and Table A.10 demonstrate that it does not

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
$l = 2 \times 10^{-3}$	39.3	78.9	90.7	76.8	0.719	0.554
$l = 5 \times 10^{-4}$	**36.1**	**75.0**	**91.9**	**78.2**	**0.736**	**0.568**
$l = 1 \times 10^{-3}$	69.3	106.6	79.3	67.4	0.588	0.468

Table A.8.: Relative error metrics for GRU with different learning rates l.

really matter if 1 s is used instead. However, using 3 s of input hurts the relative pose quality which means that long term dependencies are not relevant to the task of motion prediction. We will adopt the convention and use 2 s because the sequence lengths influences the amount of test data you get from a long video. Therefore, a consistent sequence length increases comparability of performance metrics.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
s=1	**41.5**	105.0	**90.5**	**67.0**	**0.696**	**0.454**
s=2	41.7	**104.8**	**90.5**	**67.0**	0.694	0.451
s=3	43.5	106.6	89.9	66.4	0.684	0.443

Table A.9.: Absolute error metrics for GRU with different input sequence lengths s in seconds.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
s=1	**35.8**	75.6	**91.9**	78.0	**0.738**	**0.569**
s=2	36.1	**75.0**	**91.9**	**78.2**	0.736	0.568
s=3	37.5	76.3	91.5	77.7	0.728	0.561

Table A.10.: Relative error metrics for GRU with different input sequence lengths s in seconds.

A.1.6. Output Length Schedule

In Section 4.2, we described that gradually increasing the target sequence length every ith epoch by one frame greatly helps the performance. From Table A.11 and Table A.12, we can see that both tracking and relative pose quality increase with slower schedules, i.e., schedules which train longer on each target length. The tendency stagnates around increasing the length every fourth epoch. Note

that short term predictions benefit more from $i = 4$, but long term prediction is better with $i = 3$. This is not too surprising considering that the learning rate decreases strictly monotonously so that a slower output length schedule has less time to adapt to long term predictions.
The tables do not show the performance without any length schedule but preliminary tests indicated a performance significantly worse than that for $i = 2$.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
$i = 2$	43.3	106.3	89.9	66.7	0.685	0.446
$i = 3$	41.7	104.8	90.5	**67.0**	0.694	**0.451**
$i = 4$	**41.2**	**104.7**	**90.6**	66.9	**0.697**	**0.451**

Table A.11.: Absolute error metrics for GRU with different number of epochs i until the next output length increment.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
$i = 2$	37.2	76.0	91.6	77.9	0.729	0.565
$i = 3$	36.1	**75.0**	91.9	**78.2**	0.736	**0.568**
$i = 4$	**35.8**	75.1	**92.0**	78.1	**0.738**	**0.568**

Table A.12.: Relative error metrics for GRU with different number of epochs i until the next output length increment.

A.1.7. Sample Heuristic

The last ablation concerns a heuristic mentioned in Section 6.3.2. We argued that the training phase should skip misleading motion samples where noticeable movement only occurred at the end if at all. However, Table A.13 and Table A.14 demonstrate that no advantage is gained from this filtering heuristic. For the sake of consistency, we leave the filtering heuristic enabled in later tests but discourage future research from using it since no evidence indicates its helpfulness.

A.2. Transformer

Analogue to the GRU architecture, we also evaluated the common hyperparameters for the transformer. Furthermore, some hyperparameters like the number of attention heads are analyzed as well.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
w/ heuristic	**41.7**	**104.8**	**90.5**	**67.0**	**0.694**	**0.451**
w/o heuristic	41.9	104.9	90.4	**67.0**	0.693	0.450

Table A.13.: Absolute error metrics for GRU with and without filtering idle sequences.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
w/ heuristic	**36.1**	**75.0**	**91.9**	**78.2**	**0.736**	**0.568**
w/o heuristic	36.4	75.3	91.8	78.1	0.734	**0.568**

Table A.14.: Relative error metrics for GRU with and without filtering idle sequences.

A.2.1. Absolute and Relative Loss

As we can see in Table A.15 and Table A.16, the importance of absolute and relative loss term show a similar behavior to that for the GRU architecture. As a consequence, we keep a perfect balance between absolute and relative loss.
Training results are not reproducible with newer code versions, because the initial order of the training data depended on the hardware. This experiment still uses a dropout probability of 10 % and has weight decay disabled.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
$\alpha = 0.1$	52.4	122.4	86.1	61.9	0.639	0.411
$\alpha = 0.5$	**50.6**	117.5	**86.8**	**63.2**	**0.649**	**0.422**
$\alpha = 0.9$	50.9	**117.3**	86.7	**63.2**	0.648	**0.422**

Table A.15.: Absolute error metrics for transformer with different importance α of the absolute loss term.

A.2.2. Number of Attention Heads

In multi-head attention, the number of heads defines under how many aspects two token can be similar or relevant to each other. A typical choice, e.g., in [VSP+17, ACKH20], is to use eight heads. From Table A.17, we can see that four heads are most likely not enough to accurately capture temporal dependencies. The choice between 8 and 16 heads is rather unclear as the increase in performance

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
$\alpha = 0.1$	**40.3**	**80.8**	**90.2**	**76.0**	**0.717**	**0.552**
$\alpha = 0.5$	41.8	82.7	89.6	75.4	0.709	0.546
$\alpha = 0.9$	43.2	84.7	89.1	74.7	0.701	0.539

Table A.16.: Relative error metrics for transformer with different importance α of the absolute loss term.

of 16 heads could be caused by chance. We therefore keep using 8 heads to have more comparability with previous results.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
4 heads	48.2	114.0	87.8	64.1	0.662	0.430
8 heads	47.6	112.9	88.0	64.3	0.664	0.432
16 heads	**47.3**	**112.4**	**88.2**	**64.5**	**0.666**	**0.433**

Table A.17.: Absolute error metrics for transformer with different numbers of attention heads.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
4 heads	40.1	80.8	90.2	76.1	0.719	0.554
8 heads	39.8	80.3	90.4	76.2	0.720	0.554
16 heads	**39.4**	**79.5**	**90.5**	**76.5**	**0.722**	**0.557**

Table A.18.: Relative error metrics for transformer with different numbers of attention heads.

A.2.3. Batch Size

With a default batch size of 32, our transformer has a significantly smaller batch size than the GRU. However, Table A.19 and Table A.20 do not highlight a strong influence of the batch size on the performance, which is unlike Section 8.2.5, where a batch size of 64 did not work at all. Note that the comparison does not include the GRU batch size of 128 because tests in an early stage showed a decrease in performance. These results should probably be reproduced with the current setup.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
$b = 16$	48.0	113.3	87.8	64.2	0.662	0.431
$b = 32$	**47.6**	**112.9**	**88.0**	64.3	**0.664**	0.432
$b = 64$	47.8	113.7	87.9	**64.4**	**0.664**	**0.433**

Table A.19.: Absolute error metrics for transformer with different batch sizes b.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
$b = 16$	40.4	80.8	90.1	76.1	0.717	0.553
$b = 32$	39.8	80.3	90.4	76.2	0.720	0.554
$b = 64$	**39.5**	**80.0**	**90.5**	**76.3**	**0.722**	**0.556**

Table A.20.: Relative error metrics for transformer with different batch sizes b.

A.2.4. Learning Rate

Table A.21 clearly shows that a small learning rate of 0.0005 is as good as the default learning rate of Adam. Similar to Appendix A.1.4, a higher learning rate does not lead to stable convergence. These results are reinforced by Table A.22. We therefore keep the default learning rate of 0.001.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
$l = 5 \times 10^{-4}$	**47.3**	**112.6**	**88.1**	**64.5**	**0.666**	**0.433**
$l = 1 \times 10^{-3}$	47.6	112.9	88.0	64.3	0.664	0.432
$l = 2 \times 10^{-3}$	136.7	282.5	58.1	40.5	0.404	0.273

Table A.21.: Absolute error metrics for transformer with different learning rates l.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
$l = 2 \times 10^{-3}$	**39.5**	**80.1**	**90.5**	**76.3**	**0.722**	**0.555**
$l = 5 \times 10^{-4}$	39.8	80.3	90.4	76.2	0.720	0.554
$l = 1 \times 10^{-3}$	77.0	125.3	77.3	64.1	0.581	0.455

Table A.22.: Relative error metrics for transformer with different learning rates l.

A.2.5. Loss Functions

The results for different loss functions in Table A.23 and Table A.24 are basically identical to those for the GRU in Section 8.2.6. We will therefore keep using the Manhattan-based loss.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
L1 loss	**47.6**	**112.9**	**88.0**	**64.3**	**0.664**	**0.432**
L2 loss	54.1	119.9	85.9	60.8	0.614	0.384

Table A.23.: Absolute error metrics for transformer with different loss functions.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
L1 loss	**39.8**	**80.3**	**90.4**	**76.2**	**0.720**	**0.554**
L2 loss	44.1	84.6	88.9	74.1	0.687	0.523

Table A.24.: Relative error metrics for transformer with different loss functions.

A.2.6. Length of Input

Similar to the results in Appendix A.1.5, longer inputs rather seem to hurt the performance than help it, see Table A.25 and Table A.26. We therefore stick to the convention of using 2 s of input poses prominent in literature.
An interesting difference to Appendix A.1.5 is that longer input sequences do not obstruct the transformer as much as the GRU. This is probably caused by the fact that attention works for arbitrarily distant items (within memory constraints) whereas hidden state propagation needs to balance between short and long term dependencies.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
$s = 1$	**47.3**	113.2	**88.0**	**64.6**	**0.668**	**0.437**
$s = 2$	47.6	**112.9**	**88.0**	64.3	0.664	0.432
$s = 3$	47.6	113.4	**88.0**	64.1	0.664	0.430

Table A.25.: Absolute error metrics for transformer with different input sequence lengths s in seconds.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
$s = 1$	**39.4**	**79.9**	**90.5**	**76.4**	**0.724**	**0.559**
$s = 2$	39.8	80.3	90.4	76.2	0.720	0.554
$s = 3$	39.7	80.6	90.4	76.1	0.721	0.553

Table A.26.: Relative error metrics for transformer with different input sequence lengths s in seconds.

A.2.7. Output Length Schedule

The tables in Table A.27 and Table A.28 do not indicate a clear result on what output length schedules work well for transformers. Unlike the schedules for GRUs, they do not seem to have a huge impact on the performance which is again probably related to how attention works in comparison to hidden states. We therefore conclude that the output length schedule is largely irrelevant to the performance of the transformer.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
$i = 2$	48.0	**112.5**	87.8	**64.5**	0.663	**0.432**
$i = 3$	**47.6**	112.9	**88.0**	64.3	**0.664**	**0.432**
$i = 4$	47.8	115.2	**88.0**	63.8	**0.664**	0.429

Table A.27.: Absolute error metrics for transformer with different number of epochs i until the next output length increment.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
$i = 2$	40.0	**80.0**	90.3	**76.4**	0.719	**0.555**
$i = 3$	39.8	80.3	90.4	76.2	0.720	0.554
$i = 4$	**39.6**	80.9	**90.5**	76.0	**0.722**	0.553

Table A.28.: Relative error metrics for transformer with different number of epochs i until the next output length increment.

A.2.8. Data Normalization

The results for feature-wise data normalization can be found in Table A.29 and Table A.30. Similar to GRUs, transformers seem to perform better when each

feature is centered and has unit-variance. However, the impact of normalized features is more minuscule for transformers. This is almost certainly caused by the large number of layer normalization operations in a transformer, which already ensure that the set of features are normalized.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
w/ normalize	**47.6**	**112.9**	**88.0**	**64.3**	**0.664**	**0.432**
w/o normalize	49.7	115.8	87.2	63.3	0.655	0.424

Table A.29.: Absolute error metrics for transformer with and without normalization.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
w/ normalize	**39.8**	**80.3**	**90.4**	**76.2**	**0.720**	**0.554**
w/o normalize	42.0	83.3	89.6	75.2	0.710	0.544

Table A.30.: Relative error metrics for transformer with and without normalization.

A.2.9. Sample Heuristic

The results for filtering misleading motions can be found in Table A.31 and Table A.32. They are completely analogue to those for GRUs in Appendix A.1.7, which is to say that the heuristic is not able to proof its usefulness. Again, we keep it activated for higher consistency between all experiments but do not recommend it in new experimental setups.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
w/ heuristic	**47.6**	**112.9**	**88.0**	**64.3**	**0.664**	**0.432**
w/o heuristic	47.9	113.7	87.8	64.2	0.663	0.431

Table A.31.: Absolute error metrics for transformer with and without filtering idle sequences.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
w/ heuristic	**39.8**	**80.3**	**90.4**	**76.2**	**0.720**	0.554
w/o heuristic	**39.8**	**80.3**	90.3	**76.2**	**0.720**	**0.555**

Table A.32.: Relative error metrics for transformer with and without filtering idle sequences.

A.2.10. Temporal Masking

As mentioned before, most transformers apply masked self-attention for sequence generation because the auto-regressive connection implies that future inputs reveal the solution at the current step during training. Consequently, masking is used to prevent the network from cheating by accessing information it will not have during inference. However, in our training, future poses in the input are replaced a simple heuristic which can also be applied in inference. Consequently, the masking is not necessary. Since it restricts information to only flow forward in time, it is not surprising that applying temporal masking in our case impedes the prediction, which can be seen in Table A.33 and Table A.34.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
w/ mask	51.2	117.4	86.8	63.4	0.643	0.421
w/o mask	**47.6**	**112.9**	**88.0**	**64.3**	**0.664**	**0.432**

Table A.33.: Absolute error metrics for transformer with and without temporal masking.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
w/ mask	41.6	81.6	89.8	75.8	0.707	0.548
w/o mask	**39.8**	**80.3**	**90.4**	**76.2**	**0.720**	**0.554**

Table A.34.: Relative error metrics for transformer with and without temporal masking.

A.2.11. Scaling Features

In the original transformer [VSP+17], the word embedding is scaled up by a constant factor before adding the positional encoding, however, Vaswani et al.

do not provide any motivation. Unlike a word embedding which simply clusters "similar" words, joint locations have a clear interpretable meaning and scale. Table A.36 demonstrates that it is best not to interfere with that scale. This is even more crucial for modeling global positions, see Table A.35.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
w/ scaling	51.9	118.9	86.5	62.9	0.643	0.418
w/o scaling	**47.6**	**112.9**	**88.0**	**64.3**	**0.664**	**0.432**

Table A.35.: Absolute error metrics for transformer with and without scaling up the embedding vector.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
w/ scaling	42.0	82.5	89.6	75.5	0.708	0.546
w/o scaling	**39.8**	**80.3**	**90.4**	**76.2**	**0.720**	**0.554**

Table A.36.: Relative error metrics for transformer with and without scaling up the embedding vector.

B

Further Results on Multi-Person Data

Most of the hyperparameters were tuned on AMASS validation data, but some parameters might be reconsidered on a different dataset. In the following, these parameters are analyzed for transformers and GRUs. Some of the tuning results for transformers can also be found in Section 9.1.

B.1. Transformer

Although most tuning results for transformers are presented in Section 9.1, the following hyperparameter showed little influence and is therefore presented in this appendix.

B.1.1. Filtering Threshold

As mentioned in Section 6.4.2 and Section 7.2.2, some Kinect sequences contained too many consecutive frames with poor quality which resulted in unrealistic discontinuities. We filter those sequences for the test split, but maybe those sequences should still be included in the training data. Table B.1 show that a lower threshold has a bad effect on the performance. This is obvious since the network will lack robustness against these discontinuities if it never encounters them during training. However, we can also see in Table B.2 that a higher threshold does not hinder the network, which suggests that the transformer is robust against such discontinuities. Interestingly enough, the same holds for GRUs, i.e., the discontinuity does not corrupt the hidden state (compare Appendix B). In the following experiments, we will keep the threshold at 0.4 m.

B.2. GRU

Analogously to the transformer fine-adaption on Kinect data, we also performed the same tests for our GRU.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
$t = 0.2$	38.6	56.3	91.2	84.0	0.701	0.615
$t = 0.4$	**37.5**	**54.4**	**91.6**	**84.5**	**0.704**	**0.619**
$t = 0.6$	37.8	55.1	91.5	84.3	0.702	0.615

Table B.1.: Absolute error metrics for transformer with different filter thresholds t during training.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
$t = 0.2$	27.5	39.9	94.2	89.6	0.789	0.719
$t = 0.4$	**26.6**	**38.5**	**94.4**	**90.0**	**0.792**	**0.723**
$t = 0.6$	26.7	38.9	**94.4**	89.8	0.791	0.721

Table B.2.: Relative error metrics for transformer with different filter thresholds t during training.

B.2.1. Filtering Threshold

As for transformers in Appendix B.1.1, the filtering threshold for discontinuities during training has little impact on the performance of GRUs, see Table B.3. Note in Table B.4 that large discontinuities do not corrupt the hidden state of the GRU.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
$t = 0.2$	39.5	59.5	90.9	83.6	0.696	0.611
$t = 0.4$	**38.4**	**57.3**	**91.4**	**84.2**	**0.700**	**0.615**
$t = 0.6$	**38.4**	57.7	91.3	84.0	0.699	0.613

Table B.3.: Absolute error metrics for GRU with different filter thresholds t during training.

B.2.2. Dimension of Hidden State

As in Section 9.1.1, the hidden state dimension can be chosen smaller for the Kinect datasets, see Table B.5 and Table B.6. In Section 8.2.3, a hidden state of 512 neurons was too small to accurately capture the complexity of AMASS.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
$t = 0.2$	27.3	39.4	94.4	90.1	0.789	0.720
$t = 0.4$	**26.4**	**38.1**	94.6	**90.4**	**0.791**	**0.724**
$t = 0.6$	**26.4**	**38.1**	**94.7**	**90.4**	**0.791**	**0.724**

Table B.4.: Relative error metrics for GRU with different filter thresholds t during training.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
$h = 512$	38.9	58.8	91.0	83.7	0.698	0.611
$h = 1024$	**38.4**	**57.3**	**91.4**	**84.2**	**0.700**	**0.615**

Table B.5.: Absolute error metrics for GRU with different hidden state dimensions h.

	MPJPE		PCK 10 cm		PCK AUC	
milliseconds	400	1000	400	1000	400	1000
$h = 512$	26.8	38.8	94.5	90.1	0.790	0.721
$h = 1024$	**26.4**	**38.1**	**94.6**	**90.4**	**0.791**	**0.724**

Table B.6.: Relative error metrics for GRU with different hidden state dimensions h.

Bibliography

[AAR+20] Vida Adeli, Ehsan Adeli, Ian Reid, Juan Carlos Niebles, and Hamid Rezatofighi. Socially and contextually aware human motion and pose forecasting. *IEEE Robotics and Automation Letters*, 5(4):6033–40, 2020.

[ACKH20] Emre Aksan, Peng Cao, Manuel Kaufmann, and Otmar Hilliges. A spatio-temporal transformer for 3D human motion prediction. *arXiv:2004.08692v2*, 2020.

[AGR+16] Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. Social LSTM: Human trajectory prediction in crowded spaces. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[AKH19] Emre Aksan, Manuel Kaufmann, and Otmar Hilliges. Structured prediction helps 3D human motion modelling. In *International Conference on Computer Vision*, 2019.

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag, 1 edition, 2006.

[BKH16] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *arXiv:1607.06450v1*, 2016.

[CGM+20] Zhe Cao, Hang Gao, Karttikeya Mangalam, Qizhi Cai, Minh Vo, and Jitendra Malik. Long-term human motion prediction with scene context. In *European Conference on Computer Vision*, 2020.

[CMS+20] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European Conference on Computer Vision*, 2020.

[Cro16] David F. Crouse. On implementing 2D rectangular assignment algorithms. *IEEE Transactions on Aerospace and Electronic Systems*, 52(4):1679–96, 2016.

[CSWS17] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2D pose estimation using part affinity fields. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[CvMG+14] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Empirical Methods in Natural Language Processing*, 2014.

[CZ17] João Carreira and Andrew Zisserman. Quo vadis, action recognition? A new model and the Kinetics dataset. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[DVJR19] Xiaoxiao Du, Ram Vasudevan, and Matthew Johnson-Roberson. Bio-LSTM: A biomechanically inspired recurrent neural network for 3-D pedestrian pose and gait prediction. *IEEE Robotics and Automation Letters*, 4(2):1501–8, 2019.

[FB81] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–95, 1981.

[FD87] Da-Fei Feng and Russel F. Doolittle. Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *Journal of Molecular Evolution*, 25(4):351–60, 1987.

[FLFM15] Katerina Fragkiadaki, Sergey Levine, Panna Felsen, and Jitendra Malik. Recurrent network models for human dynamics. In *International Conference on Computer Vision*, 2015.

[GB10] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics*, 2010.

[GBC16] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 1 edition, 2016.

[GBXAP21] Wen Guo, Xiaoyu Bie, and Francesc Moreno-Noguer Xavier Alameda-Pineda. Multi-person extreme motion prediction. *arXiv:2105.08825v3*, 2021.

[GSAH17] Partha Ghosh, Jie Song, Emre Aksan, and Otmar Hilliges. Learning human motion models for long-term predictions. In *International Conference on 3D Vision*, 2017.

[HGM19] Alejandro Hernandez, Jürgen Gall, and Francesc Moreno. Human motion prediction via spatio-temporal inpainting. In *International Conference on Computer Vision*, 2019.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–80, 1997.

[HZ03] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2 edition, 2003.

[IPOS14] Catalin Ionescu, Dragos Papava, Vlad Olaru, and Cristian Sminchisescu. Human3.6M: Large scale datasets and predictive methods for 3D human sensing in natural environments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(7):1325–39, 2014.

[JLT+15] Hanbyul Joo, Hao Liu, Lei Tan, Lin Gui, Bart Nabbe, Iain Matthews, Takeo Kanade, Shohei Nobuhara, and Yaser Sheikh. Panoptic Studio: A massively multiview system for social motion capture. In *International Conference on Computer Vision*, 2015.

[Kab78] Wolfgang Kabsch. A discussion of the solution for the best rotation to relate two sets of vectors. *Acta Crystallographica Section A*, 34(5):827–8, 1978.

[KB15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[KBM+20] Jogendra Nath Kundu, Himanshu Buckchash, Priyanka Mandikal, Rahul M V, Anirudh Jamkhandi, and R. Venkatesh Babu. Cross-conditioned recurrent networks for long-term synthesis of inter-person human motion interactions. In *IEEE Winter Conference on Applications of Computer Vision*, 2020.

[LH19] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

[LHL+17] Chunhui Liu, Yueyu Hu, Yanghao Li, Sijie Song, and Jiaying Liu. PKU-MMD: A large scale benchmark for skeleton-based human action understanding. In *Workshop on Visual Analysis in Smart and Connected Communities*, 2017.

[LMR+15] Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J. Black. SMPL: A skinned multi-person linear model. *ACM Transactions on Graphics*, 34(6):248:1–248:16, 2015.

[LSP+19] Jun Liu, Amir Shahroudy, Mauricio Perez, Gang Wang, Ling-Yu Duan, and Alex C. Kot. NTU RGB+D 120: A large-scale benchmark for 3D human activity understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(10):2684–701, 2019.

[LWJ+19] Zhenguang Liu, Shuang Wu, Shuyuan Jin, Qi Liu, Shijian Lu, Roger Zimmermann, and Li Cheng. Towards natural and accurate future motion prediction of humans and animals. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

[LYRK21] Ruilong Li, Shan Yang, David A. Ross, and Angjoo Kanazawa. AI choreographer: Music conditioned 3D dance generation with AIST++. In *International Conference on Computer Vision*, 2021.

[MBR17] Julieta Martinez, Michael J. Black, and Javier Romero. On human motion prediction using recurrent neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[MGT+19] Naureen Mahmood, Nima Ghorbani, Nikolaus F. Troje, Gerard Pons-Moll, and Michael J. Black. AMASS: Archive of motion capture as surface shapes. In *International Conference on Computer Vision*, 2019.

[MKLTF21] Tim Meinhardt, Alexander Kirillov, Laura Leal-Taixé, and Christoph Feichtenhofer. TrackFormer: Multi-object tracking with transformers. *arXiv:2101.02702v2*, 2021.

[MLS20] Wei Mao, Miaomiao Liu, and Mathieu Salzmann. History repeats itself: Human motion prediction via motion attention. In *European Conference on Computer Vision*, 2020.

[MLSL19] Wei Mao, Miaomiao Liu, Mathieu Salzmann, and Hongdong Li. Learning trajectory dependencies for human motion prediction. In *International Conference on Computer Vision*, 2019.

[MRC+17] Dushyant Mehta, Helge Rhodin, Dan Casas, Pascal Fua, Oleksandr Sotnychenko, Weipeng Xu, and Christian Theobalt. Monocular 3D human pose estimation in the wild using improved CNN supervision. In *International Conference on 3D Vision*, 2017.

[PGM+19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward

Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Neural Information Processing Systems*, 2019.

[Sár21] István Sárándi. Personal communication, 2021.

[SC04] Stan Salvador and Philip Chan. FastDTW: Toward accurate dynamic time warping in linear time and space. In *KDD workshop on mining temporal and sequential data*, 2004.

[SLAL21] István Sárándi, Timm Linder, Kai Oliver Arras, and Bastian Leibe. MeTRAbs: Metric-scale truncation-robust heatmaps for absolute 3D human pose estimation. *IEEE Transactions on Biometrics, Behavior, and Identity Science*, 3(1):16–30, 2021.

[SSBD14] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms.* Cambridge University Press, 1 edition, 2014.

[SSVO10] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Robotics: Modelling, Planning and Control*. Springer-Verlag, 1 edition, 2010.

[VJ01] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2001.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Neural Information Processing Systems*, 2017.

[VZ00] Yehuda Vardi and Cun-Hui Zhang. The multivariate L1-median and associated data depth. *Proceedings of the National Academy of Sciences*, 97(4):1423–6, 2000.

[WB18] Nicolai Wojke and Alex Bewley. Deep cosine metric learning for person re-identification. In *IEEE Winter Conference on Applications of Computer Vision*, 2018.

[WK20] Renjie Wu and Eamonn J. Keogh. FastDTW is approximate and generally slower than the algorithm it approximates. *IEEE Transactions on Knowledge and Data Engineering*, 2020. Early Access.

[ZBS+16] Liang Zheng, Zhi Bie, Yifan Sun, Jingdong Wang, Chi Su, Shengjin Wang, and Qi Tian. MARS: A video benchmark for large-scale person re-identification. In *European Conference on Computer Vision*, 2016.