Kinematic Constrained Loss Function for Human Motion Prediction

Terrence Thurk
Michael Henderson
David Doellstedt
Georgia Institute of Technology

tthurk3@gatech.edu
mhenderson43@gatech.edu
ddoellstedt3@gatech.edu

Abstract

Modeling human motion is a very complex task, intersecting fields of graphics, computer vision, human interaction, motion synthesis, animation, and virtual and augmented reality. In recent research [1, 2, 4, 5], deep learning models have been used to attempt to produce motion prediction for the aforementioned applications. In these applications, motion accuracy is very important in producing life-like and realistic human motion. This paper will describe several approaches to deep learning loss functions that enable greater accuracy in motion prediction tasks. In addition, we introduce a model that incorporates spatial and temporal self-attention layers, similar in concept to Aksan et al. *[5]. We trained this model with our custom loss function to achieve near state-of-the-art performance.*

1. Introduction/Background/Motivation

1.1. Introduction

Recently, large labeled motion capture datasets, such as the AMASS dataset [6] were created to provide the foundation that can be used to make pose predictions possible for human motion. AMASS consists of 48 hours of motion capture data where one pose is described in frames. Each frame captures pose as 3D angles that describe the orientation of 24 joints. Using this data, deep learning models can be trained to make human motion predictions.

1.2. Background

Our primary focus is enhancing the loss function used in previous research models. We believe that this is an overlooked area of optimization. Additionally, we would like to improve upon the baseline models using methods similar to the spatio-temporal self-attention models outlined by Aksan *et al*. [5]. To this end, we provide a brief review of related work

1.2.1 Loss Functions

Existing models have primarily used Euclidean loss or mean squared error between the predicted and ground truth poses for each joint angle component. This is true in both earlier work [1, 2] and more recent state-of-the-art models [4, 5]. The primary focus of research has been to improve the performance of the model through new architectures. MSE is an obvious choice for loss due to the ground truth and prediction being a continuously-valued tensor.

However, a simple MSE between the ground truth and predicted angles does not take into account the complexity of joint rotation and movement. Gui *et al*. [3] attempt to address some of this complexity in their research. They focus on improving the performance of an existing sequence to sequence architecture using a custom loss metric. Their loss function uses the geodesic distance, which is the shortest path in three dimensions between two rotations. In this case, the two rotations they compare are the ground truth and the prediction. The authors claim that Euclidean distance makes the loss of predictions non-smooth which can lead to discontinuities of movement in the short term and regression to the mean pose in the long term.

Although not strictly related to the loss function, the authors of several papers have used techniques or auxiliary data to improve the training performance of their models. Since their intent was to improve the training process, we will mention them here as well.

Discontinuity Several papers [2, 3, 4] describe an issue with discontinuity between the last frames of the input sequence and the first frames of the predicted sequence. The decoder developed by Li *et al*. [4] has a residual connection so that the model only predicts the difference between the previous frame and the current frame, i.e. the velocity. They claim that predicting the velocity leads to less discontinuity in the first few frames of the prediction. Gui *et al*. [3] used GANs over longer sequences to find discontinuities using adversarial training, forcing the generator to generate smoother transitions in predictions.

Pose Drift The difference between the ground truth pose and the predicted pose tends to increase with each frame [2]. To account for this drift when training, Fragkiadaki *et* *al*. [1] introduce zero mean Gaussian noise. Additionally, the authors increase the magnitude of the noise throughout training. Martinez *et al*. [2] claim that the Gaussian noise is difficult to tune, making it a suboptimal technique to reduce drift. To reduce tuning during training, for all predictions they use the previous prediction as input to the decoder for the current frame. This is opposed to typical sequence to sequence training that uses a non-zero ratio of ground truth labels as input.

In summary, most auxiliary techniques to improve training still rely on loss functions that use Euclidean distance or a similar metric. Other techniques exist to reduce drift and discontinuities, but these are either overly complex or difficult to tune. We believe that by incorporating the first and second order derivatives of predictions into the loss function, we can improve upon the performance of existing models.

1.2.2 Related Models

Recurrent Models Recurrent models have historically been the most common deep-learning architecture for motion prediction. ERD uses an encoder, followed by RNN layers, followed by a decoder. These are not sequence to sequence encoder and decoders. It is a single pass from input sequence to output sequence. As noted above, the authors add Gaussian noise to the input to simulate drift in predictions.

Martinez *et al*. [2] improve on this model by using an RNN sequence-to-sequence (seq2seq) architecture. They find superior performance when using tied weights for the encoder and decoder. As noted previously, they also use residual connections so that the model provides velocity calculations rather than position

Non-recurrent Models RNNs have several issues inherent to their architectures. For long input sequences, they have difficulty utilizing early frames from the sequence. Also, they tend to collapse to a single pose for longer prediction sequences. Others attempt to overcome these limitations using CNNs and transformer models.

Li *et al*. [4] introduce a CNN architecture that uses a convolutional layer hierarchy to find relationships between distant frames. It has separate encoders for long term and short-term relationships. Both of these data are passed to the decoder to generate the subsequent frame. Using a two CNNs allows the architecture to focus on relationships at different intervals and avoids the long-term pose. In addition, the short-term encoder is used recursively for each predicted frame, which reduces discontinuities as discussed by Martinez *et al*. [2].

Aksan*et al*. [5] use a spatio-temporal Transformer (ST-Transformer) architecture to achieve results better than previous state-of-the-art. Unlike other state-of-the-art architectures, it does not use a seq2seq architecture. One of the key contributions of the model is to apply self-attention to both the temporal data as well as the spatial data. Other models take into account relationship between frames, but Aksan *et al*. [5] also account for relationships between joints to generate a more realistic pose. It uses this to predict the subsequent pose in the sequence. Like Li *et al*. [4] and Martinez *et al*. [2], the predicted frame is used to recursively train the model to generate the subsequent frame.

In summary, we believe that spatio-temporal Transformers achieve state-of-the-art performance and should be an area of continued research. We believe that there are other architectures that can incorporate these concepts and also provide state-of-the-art performance, perhaps with less complexity than ST-Transformer.

1.3. Motivation

This project aims to augment the current/create a new metric/loss term that incorporates kinematic constraints (acceleration and velocity) across multiple frames to improve the accuracy of human motion prediction in deep learning models.

Greater accuracy in motion prediction will satisfy the need to produce realistic and human-like motion in various applications such as graphics, animation, motion synthesis, virtual/augmented reality, and object motion prediction for autonomous robotics.

2. Data

We use the AMASS dataset [6] for training and evaluation. AMASS is a large, standardized dataset containing a variety of human motions. It can be used for any research that attempts to model human motion. For our dataset, we use the joint rotation data which consists of 24 joints. The AMASS dataset contains 48 hours of motion of many different types of movements. For our model, we used 120 frame (2 second) input sequences followed by 24 frame (400 millisecond) ground truth/prediction sequences. These sequence lengths were standard lengths used by other projects. The dataset used for the model contains approximately 60,000 training, 3,000 validation and 3,000 test sequences.

Due to limitations when testing our attention architecture, we randomly sample 75% of the dataset, without replacement. All other architectures use full training dataset. The validation and test datasets are left unchanged.

We use axis angles to represent the angles of the joints. Axis angles are the angles in radians in three degrees of freedom. This means that each frame has a length of 72. This representation is the most straightforward to take derivatives of the position for velocity and acceleration. The drawback of using axis angles is that there is a discontinuity at the angle represented by $-\pi$ and π. In our datasets, we avoid this issue by excluding any sequences that where the

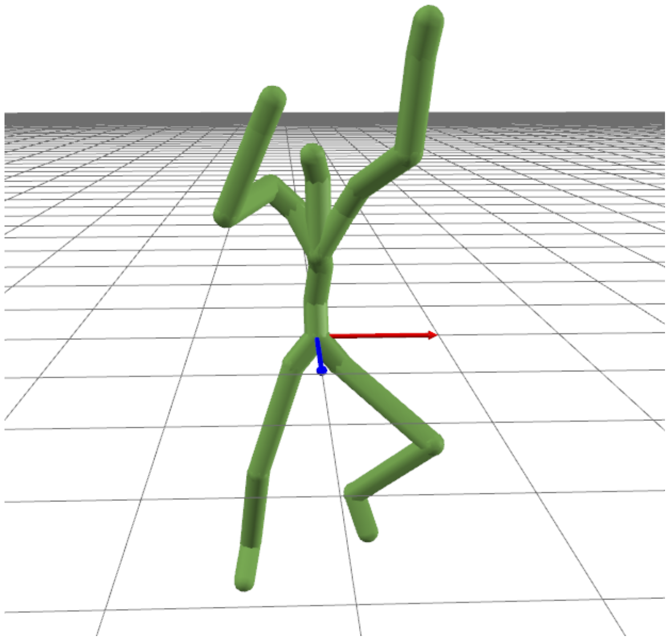


Figure 1: Visualization of a Frame of Motion Capture in the AMASS Dataset

rotation from one frame to the next is larger than four radians. This was a very small portion of the dataset - occurring in only about 0.7% of the sequences.

Alternatives to axis angles are quaternions and a rotation matrix which Aksan *et al.* [5] use for their model. The rotation matrix avoids the discontinuity of axis angles but deriving the velocity from it is a complex calculation we avoid due to time constraints. Given additional time, calculating the velocity from the rotation matrix would be our preference for working with the dataset.

Additionally, the dataset contains sequences that have rotations between frames that are too large to be valid. These are left as-is because other papers using AMASS have not mentioned cleaning the dataset. We wanted a fairer comparison to other papers using the AMASS dataset.

3. Approach

3.1. Loss Function

Prior research of deep learning approaches to human motion prediction have been focused on modifying deep learning architecture alone to achieve greater accuracy of human motion prediction [1, 2, 4, 5]. Those architectures use vanilla loss functions in training on sequences of motion capture data [1, 2, 4, 5]. The approach outlined in this paper is novel in augmenting existing deep learning loss functions with bio-mechanic constraints in the human range of motion. Two methods for this approach were analyzed.

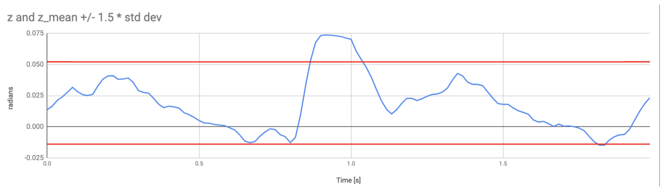


Figure 2: Angle position of the right knee joint in the z axis for one sequence, plotted along with mean+/- 1.5*standard deviation.

The first method analyzed uses mean squared error difference with the mean of position, velocity, and acceleration for each joint of each sequence of motion frames, where greater losses were assigned to motion predictions on joints where position, velocity, and accelerations contain values with greater deviation from the mean. The second method analyzed uses gating velocity and acceleration limits relative to bio-mechanical constraints of human range of motion, where motion predictions with values exceeding those limits are severely punished. This was done using an upper and lower percentile of velocity, acceleration of angular movement. Greater losses were assigned to values existing outside of the upper and lower percentile boundaries. The percentile ranges were specific to one joint and one axis of rotation. The ranges were calculated using every data point from the training set.

Figure 2 shows an example of the first method implemented on angular position for the tight knee joint in the z-axis. As you can see, angle positions for this specific sequence would be more penalized around the 1 second mark, where the values deviate farther than 1.5 * the standard deviation. This is a example from training data, but the logic is the same for generated sequences.

As an augmentation to the existing loss functions in current deep learning architectures, leveraging fundamental bio-metric constraints on human range of motion could help further enhance performance of deep learning models in generating human-like motion for prediction tasks. These methods also seek to smooth out irregularities in data that are likely caused by measurement set up and not human motion. We also believe that penalizing large velocities and accelerations will reduce discontinuities found in the first few frames of some models [2].

3.2. Model

We implemented a spatio-temporal Transformer model with self-attention for layers. Please refer to Figure 3. We apply self-attention to both spatial and temporal data. This is similar in concept to the ST-Transformer architecture discussed in Aksan *et al.* [5].

Like the ST-Transformer, we apply a self-attention network followed by a feed forward network, as discussed in

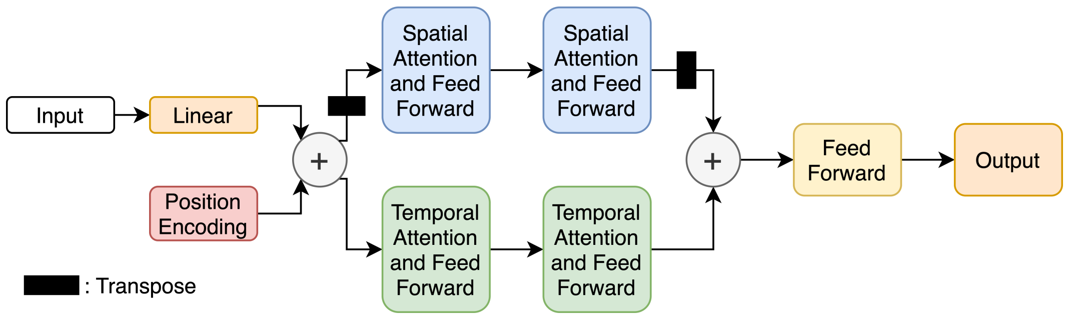

Input
Linear
Position Encoding
Spatial Attention and Feed Forward
Spatial Attention and Feed Forward
Temporal Attention and Feed Forward
Temporal Attention and Feed Forward
Feed Forward
Output
: Transpose

Figure 3: Spatio-temporal architecture

the paper Attention Is All You Need by Vaswani *et al.* [8].

Also, similar to ST-Transformer, we do not use seq2seq for our architecture. Although Aksan *et al.* [5] do not discuss their reasoning, we believe that seq2seq is not the most efficient architecture for motion prediction. It is very effective for translations between two (or more) sequences in different state spaces, such as language translation. However, that is not the task that we are addressing. We are attempting to extend an existing sequence with items from the same state space.

Embedding Our model uses a linear layer to generate embeddings from the joint positions. The model then adds a learned positional embedding to each pose in the sequence. We used a learned positional embedding rather than sinusoidal positional encoding because according to Vaswani *et al.* [8], there is no difference in performance when the input is a fixed length, as is the case with our input.

Self-Attention For temporal attention, an attention layer yields weighted sums of layers that are similar in the input sequence. For spatial attention, an attention layer yields weighted sums of poses that are similar.

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_K}})V$$

Note that we did not use a seq2seq architecture, so we did not use a mask. We used the TransformerEncoderLayer from PyTorch for the attention and feed forward networks.

For positional encoding, we transpose the sequence and joint layers and then pass it to attention and feed forward networks. The architecture of the attention layers is the same as in Vaswani *et al.* [8]. This differs from the ST-Transformer which uses shared weights between joints for the Q and V transforms. Although the positional self-attention methodology appears to be more simplistic than ST-Transformer, we saw significant improvement when applied versus positional encoding alone.

For temporal and positional encoding, we pass the data through multiple self-attention and feed forward networks before recombining the temporal and positional encoding. This also differs from ST-Transformer in that we don't recombine spatial and temporal encoding after each attention network. We believe that passing the data through multiple attention and feed forward networks before recombining allows the model to find more complex temporal and positional relationships.

Recombining Temporal and Positional Encoding We then transpose the sequence and joint layers of the positional encoding. Following this, we add the temporal and positional encodings, pass the results through another feed forward network and then a linear layer to transform the positional embedding back to the original joint space.

Note that the final feed forward network is feed forward portion of the PyTorch TransformerEncoderLayer.

Due to training time and resource constraints, our model generates all output frames at the same time, rather than one frame at a time.

Also, do to time and resource constraints, we only use an architecture with two transformer and feed-forward layers, for both spatial and temporal data.

4. Problems Encountered

The methods used in this paper utilized the Fairmotion Github repository [7] created by Facebook research. Using this repository as an upstream, code changes were made in a Georgia Tech Github repository to test the two approaches outlined above.

The first approach, which penalized joint kinematic motion (position, velocity, acceleration) deviations from the calculated mean of the respective joint by a threshold, ended up not working very well. The implementation of this extra loss made the results worse than the baseline. This decline in performance may be due to the complexity and many possible combinations of human motions that would incorrectly punish the model learning a naive deviation from the mean of a difficult task. The second approach, which was similar to the first, except a percentile was calculated instead of means/standard deviation, performed very well. A noticeable improvement in the results was observed upon implementation. The theory behind penalizing outliers in the data is to clean/sanitize irregularities in the raw data. So a balance must be found when choosing the thresholds of when to start penalizing frame measurements.

Furthermore, for both approaches, it was found that using these extra loss methods were ineffective for the angle position. Improvements were only witnessed with the velocity and acceleration aggregate filters. This makes sense because of the choice of angle measurements that were used. The training was done using the axis-angle scheme which provides the angular motion at all 72 measurements (24 joints in x, y, z) within a range of $[-\pi, \pi]$. This poses an interesting problem. When an angle makes a transition between the bounds from π to $-\pi$, or vice versa, it appears to be a massive jump in angle position, with the numbers showing about a 2π change. However, in reality, the motion of the joint is actually very small. To be able to detect these transitions, a difference in angle position between frames was calculated, which is the joints angular velocity. To mitigate the jump between the boundaries, a threshold was applied to both velocity and acceleration, to specify which frames to not include in the loss calculation. Specifically, a threshold of 4 [radiams/frame] and 4 [radians/frame^2] were chosen for velocity and acceleration calculations respectively. Anything above this threshold is considered to be a boundary jump, and not included. Angle position was not incorporated in the extra loss functions for both approaches due to the inability for the naive aggregate statistics to pick up on near boundary measurements.

Initially, we implemented a Transformer/Attention model that only had temporal attention. We found that the results of this model were significantly worse than the seq2seq model in the fairmotion project. This led us to do additional research on the spatio-temporal model and implement a similar architecture. With the improved architecture, we were able to easily outperform the seq2seq network.

5. Experiments and Results

5.1. Loss Function

To start the experiment, a baseline was created. To get the best baseline, models from the Facebook Fairmotion Github repository [7] were used. Specifically, the models considered were Sequence to Sequence (seq2seq), transformer encoder, and vanilla RNN. The loss functions considered were L1, smooth L1, and mean squared error. The optimizers considered were Noam (from Attention is All You Need [8]), Adam, and vanilla Stochastic Gradient Descent. The models were trained with a varying number of epochs, batch sizes, number of layers, and hidden dimensions. In total, 35 models with varying hyperparamters were chosen to be the baseline. Shown in Table 1 are the top 3 baseline models chosen with their respective hyperparamters. They were evaluated with the validation data after the final epoch in the form of mean angle error at 6,12,18,and 24 frames predicted into the future. The validation error was used as a metric to assess baseline performance and not the test error, to avoid overfitting models to the test set.

With the baselines chosen, the augmented loss function was created with the two approaches of using the training data to generate aggregate statistics for penalizing ”outlier” data. As referenced before, the two approaches used mean/standard deviations (deviation loss) and percentile bounds (outlier loss). Statistics were calculated on every joint, every axis, for position, velocity, and acceleration. These extra losses were summed with the mean squared error loss function. Pretty quickly, it was apparent that the deviation loss made results worse and the outlier loss improved results. See row 1 and row 2 in Table 2 for the comparison between both approaches for the augmented loss function. Row 3 shows the best model and tuned hyperparameters for the custom loss function.

5.2. Model

Due to our spatio-temporal model being a stretch goal, we are not able to test all permutations of all hyperparameters. We were able to achieve near state-of-the-art results with two layers of attention, so we did not test with more than two layers. We tested optimizers, number of attention heads, and hidden size. We used a batch size of 128 because that was what we used for the final version of the seq2seq model. A larger batch size also reduced variance when training. Our models used no learned parameters, a full training loop was done on the model with all parameters.

Optimizer We tested the Noam [8] and Adam optimizers. In our testing, the Noam optimizer had a higher variance than Adam, both in validation loss compared to training loss and validation loss between epochs. This was unexpected because the Noam optimizer was designed to reduce variance in self-attention networks. See Figure 4 for loss curves. Given this, even though the Noam loss is slightly less than Adam, we decided to use Adam for our testing.

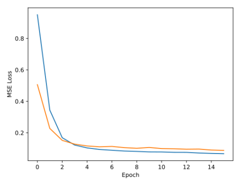
Hyperparameter Tuning We tested hidden sizes 64, 128 and 256. To reduce the number of hyperparameters, we set the feed forward dimension to twice the hidden size. With each of these hidden sizes, we tested 2, 4 and 8 attention heads. Increasing hidden size had a larger effect than number of heads. The best performance was with a hidden size of 256 and 4 attention heads. With a hidden size of 256 and 8 attention heads, the model began to overfit. Due to resource constraints, we could not test a hidden size larger than 256. The model showed a 20% improvement

Model	Loss function	optimizer	Hidden dim	Batch size	Epochs	Validation MAE (at 6, 12, 18, 24 frames)			
Seq2Seq	MSE	noam	1024	32	16	2.92	7.51	13.39	20.25
Seq2Seq	MSE	noam	1200	64	16	2.99	7.56	13.42	20.19
Transformer encoder	MSE	noam	1024	64	16	2.99	7.76	13.95	21.08
Self Attention	MSE	Adam	256/4	128	16	2.39	6.00	10.86	16.67
Self Attention (Test)	MSE	Adam	256/4	128	40	1.58	4.33	8.11	12.78

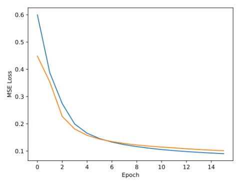
Table 1: Top 3 baselines chosen with hyperparameters. The final model was run against the test dataset. Other models were run against the validation dataset

Model	Custom loss type	Custom loss	kinematic constraint(s)	Test MAE (at 6, 12, 18, 24 frames)			
Seq2Seq	deviation	0.1	velocity	12.92	25.80	38.66	51.52
Seq2Seq	outlier	0.1	velocity	2.72	6.79	11.99	18.10
Self Attention	outlier	0.1, 0.1	velocity, acceleration	1.57	4.23	7.89	12.37

Table 2: First two rows show the comparison between the deviation and outlier loss. Outlier almost always outperformed by an order of magnitude. The third row shows the best performing configuration from all the runs.



(a) Noam



(b) Adam

Figure 4: Loss curves for Noam and Adam optimizers with a hidden size of 128 and two attention heads per layer. Blue line is training loss and red line is validation loss

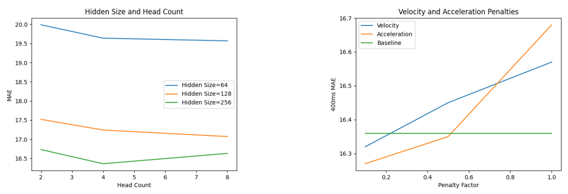


Figure 5: Mean absolute error of ground truth vs prediction at 400ms (24th frame), from the validation dataset. The first plot shows curves from tuning hidden size and number of self-attention heads. The second plot shows velocity and acceleration outliers, and various multipliers.

over seq2seq, as shown in 1

Kinematic loss After optimizing the model, we continued training with our custom loss kinematic outlier loss. We tuned the comparison type and multiplier of our loss function to further optimize the performance. With our tuned model, we were able to achieve near state-of-the-art performance, and outperformed the two layer ST-Transformer [5]. Due to different loss spaces, validation loss could not be directly compared for tuning the loss factors. Using kinematic loss, the model showed approximately a 3% improvement after frame 6, as shown in 2. See Figure 5 for the tuning results.

6. Future Work

Improvements in accuracy of motion prediction advances realistic and human-like motion generation for graphics, animation, motion synthesis, virtual/augmented reality. It is clear that creating specialized loss functions using velocity and acceleration limits relative to bio-mechanical constraints of human range of motion improves the accuracy of human motion prediction. Future work should incorporate more complex bio-mechanical constraints that emulate motion synergy and muscle movement across multiple joints. It would also be advantageous to adapt this loss function to a more robust angle schema than than the axis angle. As stated before, axis angles cause discontinuities when the joint angles crosses the $\pi/-\pi$ boundary. A custom loss function with kinematic constraints could be more robust using a rotation matrix schema for joint position.

With additional resources and time, the spatio-temporal architecture can be tested with more self-attention layers and a larger number of hyperparameters. In this architecture, the spatial and temporal data can pass through a different number of self-attention layers, depending on the complexity of the data.

References

[1] Katerina Fragkiadaki, Sergey Levine, Panna Felsen, and Jitendra Malik. Recurrent network models for human dynamics. https://arxiv.org/pdf/1508.00271.pdf 1, 2, 3

[2] Julieta Martinez, Michael J. Black, and Javier Romero. On human motion prediction using recurrent neural networks. https://arxiv.org/pdf/1705.02445.pdf 1, 2, 3

[3] Liang-Yan Gui, Yu-Xiong Wang, Xiaodan Liang, and Jose M. F. Moura. Adversarial Geometry-Aware Human Motion Prediction. https://openaccess.thecvf.com/content_ECCV_2018/papers/Liangyan_Gui_Adversarial_Geometry-Aware_Human_ECCV_2018_paper.pdf 1

[4] Chen Li, Zhen Zhang, Wee Sun Lee, Gim Hee Lee. Convolutional Sequence to Sequence Model for Human Dynamics. https://arxiv.org/pdf/1805.00655.pdf 1, 2, 3

[5] Emre Aksan, Peng Cao, Manuel Kaufmann, Otmar Hilliges. A Spatio-temporal Transformer for 3D Human Motion Prediction. https://arxiv.org/pdf/2004.08692.pdf 1, 2, 3, 4, 6

[6] Mahmood, Naureen and Ghorbani, Nima and Troje, Nikolaus F. and Pons-Moll, Gerard and Black, Michael J. AMASS: Archive of Motion Capture as Surface Shapes. https://amass.is.tue.mpg.de/ 1, 2

[7] Gopinath, Deepak and Won, Jungdam. fairmotion - Tools to load, process and visualize motion capture data. https://github.com/facebookresearch/fairmotion 4, 5

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. https://arxiv.org/pdf/1706.03762.pdf

4, 5

Student Name	Contributed Aspects
David Doellstedt	Generating metrics for MSE Deviation loss function implementation. Colab and Github project setup and maintenance. Training models and comparing performance. Final report.
Mike Henderson	Compiled relevant research from existing papers. Initial work on loss function implementation. Attention models implementation and testing (attention.py, st$_attention.py).Reportsections1.2, 2, 3.2, 5.2.$
Terrence Thurk	Preprocessing Data on Velocity/Acceleration Gating. Editing code to allow multiple training on multiple loss functions. Training models and comparing performance. Final report.

Table 3: Contributions of team members.