Motion Prediction
The Admissible Heuristics: CS 7643

Crutchlow, Sean scrutchlow3@gatech.edu
Goyal, Ankit agoyal35@gatech.edu
Lam, Jackie jlam61@gatech.edu
Xu, Chengyin cxu371@gatech.edu

Georgia Institute of Technology
North Ave NW, Atlanta, GA 30332

Abstract

Motion prediction is a subset of perception, and presents itself in many different domains. Some of which can be automotive, robotics, and many others. In order to predict motion, human in this case, spatial and temporal information must be used in order to model sequential joint motion. Previous works in motion prediction were constrained to smaller prediction horizons due to the nature of their architecture. With newer architectures such as spatio-temporal transformers, predictions over a larger horizon can be performed provided an initial sequence of motions.

1. Introduction/Background/Motivation

Model architectures such as RNNs, Seq2Seq, and transformers have been widely used for human motion prediction. Liu et. al [4] used adaptable RNNs in combination with modified Kalman filters for predicting motion of a human wrist. The model demonstrated produced lower prediction errors when compared to typical RNN or Seq2Seq models. Other RNN modifications such as Encoder Recurrent Decoder [6] have been reported in literature predicting temporal dynamics of 3D human motion. This model can predict periodic motions, but lacks in its prediction ability for those non-periodic. Existing deep RNN models have been found to suffer from inability to accurately predict long term motion [6]. Models such as LSTM-3LR and ERD were reported to accumulate error over time and as a result could not generalize well for long-term motion prediction [2].

In an attempt to overcome the pitfalls of Recurrent models, we turned towards Spatio-Temporal Transformer Network (STTN). Xu. et. al [10] reports use of STTN in long-term traffic flow prediction. The model has the ability to overcome previously mentioned pitfalls by leveraging a self-attention mechanism. The model begins by learning various human joint angle embeddings. The attention

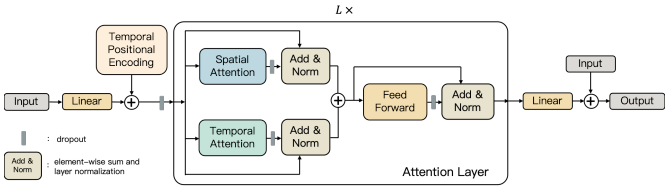

Figure 1. Spatio-temporal transformer architecture.

block defines inter-dependencies between joints, composing a joint graph, otherwise viewed as the skeletal structure, and uses these dependencies for future pose prediction. [1], [9] The goal of this project is to implement the STTN model outlined in the paper by Aksan et. al [1] from scratch.

If our STTN architecture approaches SOTA performance for predicting motions, that means the model is not just a novelty, but can be deployed in an application of choice. As previously mentioned, human motion prediction is commonly used in robotics and automotive domains. A use case in the automotive field is predicting pedestrian motion along sidewalks or crosswalks in order to know if a pedestrian will become an occlusion for an ego-vehicle trajectory.

In order to achieve SOTA performance, the architecture implemented accounts for spatial and temporal changes in human motion. This translates to the model understanding where the human skeletal structure will move in space, and how long it will take over time to do so. This understanding of human joint motion extends to s seconds in the future. If an architecture had no understanding of human joint articulation in terms of rotation or how fast they are able to move, the predicted motion may look reminiscent of rag-doll physics from a video game. Being able to capture natural joint motion through an architecture is a powerful capability.

1.1. Data

The model was studied using the AMASS dataset[5]. We chose this dataset because it is large and diverse, as it uni-

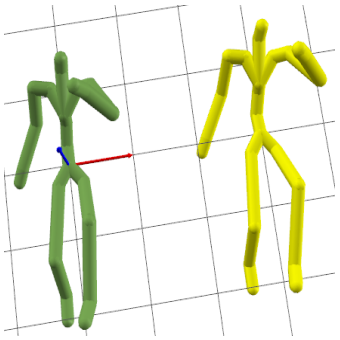

Figure 2. Visualizing predicted and reference motions for skeletal joints.

fied 15 different human motion datasets with a common framework and parameterization. The dataset consists of 42 hours of recorded human motion data capturing pose information for various joints in the human skeletal structure. The input sequence is composed of 120 poses, and the predicted output sequence is 24 poses. The dataset was available for download as a series of pickle files and has already been split into train, test and validation sets. The only preprocessing required was representing the joints with an angle representation. We explored two representations: Euler and quaternions. The benefit to using Euler angles instead of quaternions is a reduction in an extra component which in turn reduces the number of trainable model parameters. Since motion prediction takes place in 3D space, roll, pitch, and yaw are used for the Euler angles. Adding an extra component for a quaternion, x, y, z, and w, will add additional information to avoid gimbal lock. Due to the scope of this project and time constraints, the exploration of seeing if the human joints can encounter gimbal lock was not performed, and instead Euler angles were used to reduce the number of parameters in the STTN architecture.

1.2. Code Repositories

The starter code for this project was taken from Facebook's Fairmotion [3] project. The Facebook project used the PyTorch framework, and had existing baseline model architectures: RNN, Seq2Seq, and Transformer-Encoder. These models were used as baselines to compare against our own architecture for the project. In addition to the models, an error metric, mean-angle error (MAE) was provided to evaluate prediction and reference joint motions. For the AMASS dataset, subsets of it were taken to generate small, medium, and large datasets. The preprocessing script to convert raw data into axis angle rotation representation, and a custom SGD optimizer were also provided. The following changes were made to the repository:

1. Spatio Temporal Model implemented from scratch
2. Updated data loading, training, and evaluation to accommodate alternative training scheme 2.3.1
3. Dynamic Time Warping Evaluation Metric

1.3. Evaluation

Our goal was to compare our implemented STTN model to Fairmotion project baseline architectures RNN, Seq2Seq, and transformer-encoder with respect to joint error. Due to time constraints, the baseline model hyper-parameters were not tuned, but the STTN was tuned in order to get closer to SOTA performance. We looked further into different error metrics to study the training performance. The goal was to pick up the training error metrics that most realistically represented error in human motion. The key to choosing the correct error metric was to analyze the correct set of joints and then calculate the difference between their poses. The joints analyzed were ones defined as "major joints" per the AMASS DIP dataset, similar to the approach taken by the Fairmotion project from Facebook Research. A metric that was added to improve on the joint error calculation was dynamic time warping (DTW). DTW is a metric for time series classification tasks and has been previously used in both speech recognition and motion prediction [7] [8]. It calculates the temporal similarity between two subjects even though they may be traversing at a different velocity or acceleration [8]. This allows for a non-linear mapping between the prediction of skeletal joint motions if one time series for joint predictions has a lead or a lag over the other.

2. Approach

2.1. Baseline Architectures

The baseline architectures that were provided in the Facebook Fairmotion project were recurrent neural network (RNN), sequence-to-sequence (Seq2Seq), and transformer-encoder. While RNN and Seq2Seq are able to predict short sequences, they do not have good long term memory and were not expected to perform well during the entirety of the 24 frame sequence (0.4 seconds in total) for motion prediction. The transformer-encoder despite having a better temporal representation, does not have the spatial component that is needed for joint motion prediction.

2.2. Spatio-Temporal Transformer

We implemented the STTN model based on Aksan et al. [1]. So far there has not been publicly available code for this particular model. It is different from the other baseline RNN and seq2seq models because it decouples the temporal and spatial aspects of human motion by having a self-attention mechanism for each of the two aspects. This enables the model to remember motion in the past and capture spatio-temporal interactions explicitly. For every joint we define temporal attention across time and define spatial attention

over joints for a particular time step. Both the temporal and spatial attention layers are ran concurrently before being added together. The detailed components of the architecture are shown below.

2.2.1 Embedding Layer + Positional Encoding

First, the input sequence X passes through a hidden layer for each joint. Its dimensionality is $T \times N \times M$ where T represents the length of sequence, N represents the number of joints, and M represents the dimension of a joint. In this case, M=3 because we have preprocessed the raw data to form axis-angle rotation representation for each joint. The output embedding E has a dimension of $T \times N \times D$. A sinusoidal positional encoding is then added to this output embedding before going through a dropout layer and being passed to L attention layers.

2.2.2 Temporal Attention Layer

For the temporal attention layer, each joint embedding is updated using the previous time steps from the same joint. For each joint, we calculate the query, Q, key, K and value, V by passing the embeddings of the joint through a hidden layer. We also create a 2D mask, M, which prevents the model from looking at future values for its prediction. We then use Q, K, V and M to calculate the temporal attention using the formula proposed by Vaswan et. al [9]. Since we are using multi-head, we calculate Q, K, V and the resultant temporal attention independently for each head before concatenating them together and passing the result through a linear layer to form the temporal summary. The temporal summary then goes through a dropout layer. Residual connection and layer normalization are applied to get the final temporary summary of dimension $T \times N \times D$.

2.2.3 Spatial Attention Layer

In the vanilla transformer, the attention layer captures the interactions between the elements implicitly since it operates on the entire input directly. However, in our case, we introduce an additional spatial attention layer to learn the interactions between joints. For each frame, attention is applied spatially across all the joints. This is different from temporal attention where attention is applied temporally across all the frames for each joint. Unlike temporal attention, we do not need a 2D mask because spatial attention only calculates at each frame and does not have the ability to look beyond at future values. Similarly, we apply multi-head attention, dropout layer, residual connection, and layer normalization to get the final spatial summary of dimension $T \times N \times D$.

2.2.4 Feed-Forward Layer

We pass the sum of the temporal and spatial summaries into a hidden layer, a ReLU layer, and another hidden layer. Then it goes through a dropout layer followed by residual connection and layer normalization to arrive at the joint embedding output of dimension $T \times N \times D$. We stack L such attention layers to successively update the joint embeddings.

2.2.5 Final Layer

After L attention layers, the final hidden layer will project the joint embeddings from $T \times N \times D$ back to $T \times N \times M$. Each joint will have its own hidden layer. A residual connection is applied before outputting the final joint embedding predictions.

2.2.6 Loss

The loss function used by Aksan et al. [1] to train their STTN was per joint loss. This is calculated by summing up the Mean Squared Error across all the joints over all the frames 3. Through our experiments we found that using

$$L(X, \hat{X}) = \sum_{t=2}^{T+1} \sum_{n=1}^{N} \|j_t^{(n)} - \hat{j}_t^{(n)}\|_2$$

Figure 3. Per Joint Loss

per-joint loss was not computationally efficient. We then decided to use Mean Squared Error on the output directly instead of breaking it down into joints. This is a consistent approach on loss with our our baseline models.

2.2.7 Optimizer

We made use of an SGD optimizer with an initial learning rate=0.1. There is a learning rate scheduler which reduces the learning rate by $factor = 0.5$ and with $patience = 5$. Patience refers to the number of epochs with no improvement after which learning rate will be reduced. The learning rate will be reduced when the loss stops decreasing.

2.3. Challenges

The biggest challenge we faced was memory constraints and lack of computational resources while training the STTN model. Using a single NVIDIA RTX 3070, we were limited to a batch size of 32. Given the many parallel layers of this model, it would have benefited from distributed training across multiple GPUs. During hyperparameter tuning, this batch size had to be decreased when other layers such as self-attention blocks, number of attention heads, or the joint embedding size were increased thus making the

training process even slower. We highlight some of the mitigation methods for these issues below.

2.3.1 Alternative Training and Evaluation Approach

With the AMASS dataset, the source is 120 frames and the target is 24. A single target frame p_t is predicted using frames $[p_0, \cdots, p_{t-1}]$. An alternative training approach was taken, transforming the target to be the source shifted to the right by a single timestep. For example, if the input source was $[x_0, \cdots, x_{119}]$ the target would be $[x_1 \cdots x_{120}]$. Both the source and target are now 120 frames. For each original source and target pair, we now have 24 pairs, one for each frame in the original target. For the final prediction, only the final frame of the new targets are used, however, for computing loss, the full sequence is used. This simplifies the computational graph which helps mitigate potential out of memory issues. This approach circumvents out-of-memory at the expense of multiplying the number of training sequences by a factor of 24, increasing training time.

2.3.2 Use of Pytorch's Multihead Attention Module

Initially we built our multihead attention layers from scratch. This proved to be inefficient. We then decided to try out Pytorch's multihead attention module which decreased our runtime by more than 10%.

2.3.3 Reduction of the Number of Model Parameters

Initially we implemented a different weight for each joint at each time frame for the spatial query. This is according to the original paper. However, this made computation slower. We then decided to reduce the number of parameters by sharing a weight across all joints for each time frame for the spatial query. This is consistent with the approach towards temporal query stated in the original paper, where a weight is shared across all time frames for each joint.

3. Experiments and Results

All experiments were performed using the *small* subset of the AMASS dataset. The small dataset has been split into training, testing and validation with 728, 32, and 38 examples respectively. While the baseline models were trained on both the large and small datasets, only the small dataset was used when comparing to the STTN architecture. Some of the motions that the small dataset was composed of were walking, sitting, standing, jumping, crouching, and a few others.

3.1. Quantitative Evaluation

Evaluation was performed using prediction and reference joints over the sequence of predicted frames. Two error metrics were used for this, one short-term using a frame-by-frame comparison, and the other a long-term comparison capturing sequence error.

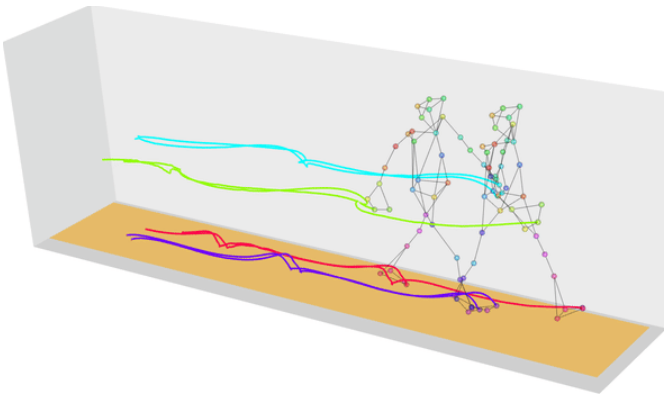

Figure 4. Dynamic time warping for predicted and reference motions.

3.1.1 Frame by Frame Error

Short-term error was computed using mean-angle error (MAE). The difference between prediction and reference joint Euler angles was taken, the mean across all joints for that single frame in the sequence was taken.

3.1.2 Sequence Error

Long-term error was computed using a combination of MAE with dynamic time warping (DTW). MAE DTW allows a non-linear mapping between joints over the sequence of predicted motions. This way the error metric MAE can be applied more appropriately if the motion predictions from the model has a lead or lag over the reference joint motions.

3.2. Results

Motion prediction exploration was successful for the scope of this paper. Considering compute resources and training time, the results of STTN over fewer epochs were comparable to that of RNN, Seq2Seq, and transformer-encoder architectures over 100 epochs.

The process of hyperparameter tuning for the STTN was constrained by the duration of computation time, and space complexity allowed on our NVIDIA RTX 3070 GPU (being 8GB). An iterative approach was used in modifying the number of attention blocks, joint embedding size, and feed-forward size. When a hyperparameter was halved or doubled, the batch size needed to be modified accordingly to allow the model to fit on the GPU. Since we were using a single GPU for training, we could not effectively parallelize the multi-head attention. As a result, we used only 2

Number Epochs	Batch Size	Number Attention Blocks	Joint Embedding Size	Number Attention Heads	Feed-Forward Size	Dropout	Optimizer	Learning Rate	Training Loss	Validation Loss	MAE
5	16	6	24	2	128	0.1	SGD	0.1	0.008 901	0.006 509	24.2522
15	32	3	32	2	64	0.1	SGD	0.1	0.004 26	0.003 373	25.6406
5	16	6	32	2	64	0.1	SGD	0.1	0.008 784	0.006 474	26.6566
3	32	3	24	2	128	0.1	SGD	0.1	0.006 538	0.003 955	28.5365

Table 1. Spatial-Temporal Transformer hyperparameter tuning using small dataset and a NVIDIA RTX 3070.

Architecture	Number Parameters	Number Epochs	Training Time [H:M:S]	Training Loss	Validation Loss	MAE	MAE DTW
RNN	4,571,208	100	0:25:05	0.006 285	0.007 373	38.045 779	39.219 787
Seq2Seq	9,068,616	100	0:21:58	0.002 33	0.004 299	27.042 418	32.644 475
Transformer-Encoder	10,947,656	100	0:25:20	0.004 731	0.005 881	31.560 428	35.579 512
Spatio-Temporal Transformer	615,248	5	19:35:58	0.008 901	0.006 509	24.252 204	30.749 429

Table 2. Predicted Motion Results for Small Dataset using a NVIDIA RTX 3070.

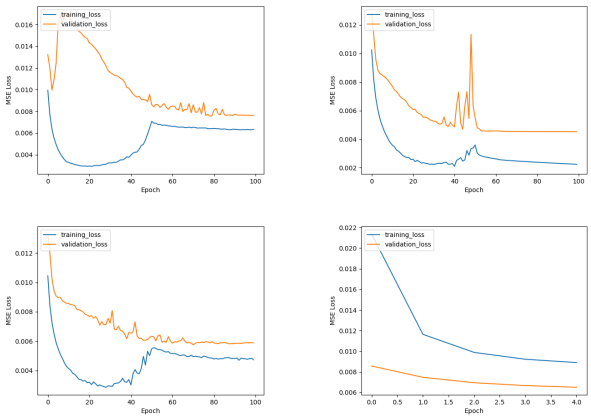

Figure 5. Losses for (top-left) RNN (top-right) Seq2Seq (bottom-left) Transformer-Encoder (bottom-right) Spatio-Temporal Transformer

attention heads, as it is the minimum number of heads for multi-head attention.

Tuning a few sets of hyperparameters, we found the best performing configuration (Table 1) is when epochs=5, batch size=16, number of attention blocks=6, joint embedding size=24, number of attention heads=2, and feed-forward size=128. Even though the training loss (0.008901) and validation loss (0.006509) are not the lowest, its MAE of 24.2522 on the test set is the lowest amongst all the configurations of the STTN and baseline models (Table 2). Since this model was only trained for 5 epochs, with additional epochs, this model should continue to improve, as it shows that it is able to generalize better with no overfitting. From the loss curves in Figure 5 we see that overfitting happens to the baseline models after around 20 epochs as training loss continues to decrease but validation loss increases.

We found that the number of attention blocks and the feed-forward size have a significant impact on performance. By increasing the number of attention blocks from 3 to 6, we increased the times the spatial and temporal summaries get updated. This should allow the model to better represent the data in both the spatial and temporal dimensions. By increasing the size of the feed-forward layer from 64 to 128, we increase the model's capacity to represent the temporal and spatial summaries. Even though the joint embedding size is decreased from 32 to 24, the MAE still decreased from over 25 to 24.2522. This shows that the joints do not require large embeddings to encode the necessary information.

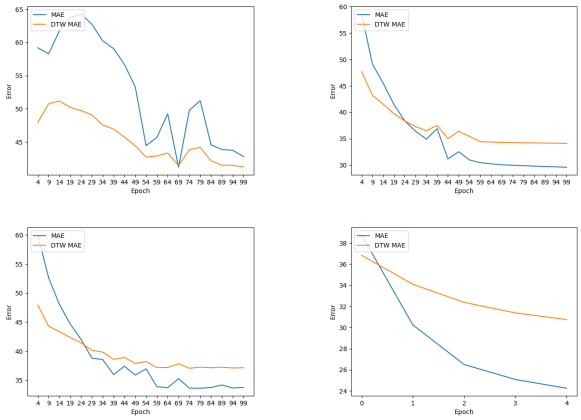

Figure 6. Error metrics for (top-left) RNN (top-right) Seq2Seq (bottom-left) Transformer-Encoder (bottom-right) Spatio-Temporal Transformer

4. Conclusion

We were able to successfully implement a STTN from scratch and implement our training scheme 2.3.1 into the Fairmotion repository. In addition, dynamic time warping was added as an additional evaluation metric. In our experiments, we were able to beat the baseline RNN, Seq2Seq, and Transformer Encoder model from the Fairmotion Repository. We found that our STTN model had lower MAE and MAE DTW errors after 5 epochs as compared to other baseline models which were trained with 100 epochs. 2. A shortcoming we ran into was not being able to optimize the code for our STTN implementation, causing excessively long training times. As a result, we were only able to perform minimal hyperparameter tuning.

References

[1] Emre Aksan, Peng Cao, Manuel Kaufmann, and Otmar Hilliges. Attention, please: A spatio-temporal transformer for 3d human motion prediction. *arXiv preprint arXiv:2004.08692*, 2, 2020. 1, 2, 3

[2] Katerina Fragkiadaki, Sergey Levine, and Jitendra Malik. Recurrent network models for kinematic tracking. *CoRR, abs/1508.00271*, 1(2):4, 2015. 1

[3] Deepak Gopinath and Jungdam Won. fairmotion - tools to load, process and visualize motion capture data. Github, 2020. 2

[4] Ruixuan Liu and Changliu Liu. Human motion prediction using adaptable recurrent neural networks and inverse kinematics. *IEEE Control Systems Letters*, 5(5):1651–1656, 2020. 1

[5] 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*, pages 5442–5451, Oct. 2019. 1

[6] Julieta Martinez, Michael J Black, and Javier Romero. On human motion prediction using recurrent neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2891–2900, 2017. 1

[7] Pascal Schneider, Raphael Memmesheimer, Ivanna Kramer, and Dietrich Paulus. Gesture recognition in rgb videos using human body keypoints and dynamic time warping. In *Robot World Cup*, pages 281–293. Springer, 2019. 2

[8] Anna Sebernegg, Peter Kán, and Hannes Kaufmann. Motion similarity modeling–a state of the art report. *arXiv preprint arXiv:2008.05872*, 2020. 2

[9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017. 1, 3

[10] Mingxing Xu, Wenrui Dai, Chunmiao Liu, Xing Gao, Weiyao Lin, Guo-Jun Qi, and Hongkai Xiong. Spatial-temporal transformer networks for traffic flow forecasting. *arXiv preprint arXiv:2001.02908*, 2020. 1

5. Appendix

- Github Repo
- Github Repo Zipped

Architecture	Number Parameters	Number Epochs	Training Time [H:M:S]	Training Loss	Validation Loss	MAE	MAE DTW
RNN	4,571,208	100	7:23:21	0.006 275	0.004 546	28.207 962	33.013 988
Seq2Seq	9,068,616	100	6:29:16	0.001 142	0.001 775	16.403 04	24.274 885
Transformer-Encoder	10,947,656	100	7:37:32	0.002 139	0.002 175	18.729 969	26.116 313

Table 3. Predicted Motion Results for Large Dataset using a NVIDIA RTX 3070.

5.1. Work Division

Student Name	Contributed Aspects	Details
Sean Crutchlow	Data Preprocessing, Model Training, Model Evaluation w.r.t plots and motion visualization, Additional Error Metrics and Data Logging	Implemented functionality to generate small, medium, large, and complete datasets. Pre-processed all dataset sizes and made available to team. Trained and tested all models evaluated and generated loss/error curves and predicted motion sequences which were made available to team. Researched additional error metrics and implemented dynamic time warping metric.
Ankit Goyal	Implementation and Analysis	Incorporated anchors for addition of STTN model to existing fairmotion code
Jackie Lam	Implementation of training scheme	Data Preprocessing wrt to training, custom loss function, training loop
Chengyin Xu	Implementation of Spatio-Temporal model	Implemented the model from scratch. Optimized the model by trying out different implementations (e.g. Pytorch multihead attention module vs implementing myself, reducing number of parameters). Discussed with Jackie on how to modify the training scheme.

Table 4. Contributions of team members.