Human Motion Prediction

Vincent Jung
vijung@student.ethz.ch

Tommaso Macrì
macrit@student.ethz.ch

András Geiszl
ageiszl@student.ethz.ch

ABSTRACT

Human motion prediction is a fundamental computer vision problem, which deeply influences our abilities to extend computers' reach to human applications. Solving it to a confidently high degree would be beneficial for many fields related to human-computer interactions (HCI). In this paper, we propose to use a Graph Convolutional network for human motion prediction and we explore two different attention networks. We see that the proposed methods achieve high accuracy on unseen motion sequences with great continuity in the predicted motion.

1 INTRODUCTION

In this paper, we will be using a Graph Convolutional Network (GCN) for human motion prediction along with a motion attention network. We will be exploring two different attention network, the first one is the same as in [5], and the second one is a novel approach that we have created. The former approach uses a simple two-layer 1D convolution on the coefficients of the human body representation through time to create feature representations and compare those to compute the attention values. The latter approach uses a Graph Convolutional Network to encode and decode human motion sequences and uses the latent variable representation created by this network to compute the values for our attention network. We thought using a GCN would be more appropriate since it could exploit the graph structure of the human body. We will see that this approach nears the performance of the convolutional one and opens the door for more exploration.

2 RELATED WORK

For solving the problem of Human Motion Prediction, various and different approaches have been used in the literature. Previously, model- and physics-based approaches have been investigated, by incorporating previous knowledge of the human kinematic chain in the predictor. Due to the high dimensionality of the problem, and to the non-linear dynamics and stochastic nature of human movements, these approaches have been proved to be partially successful only for specific types of movements. Traditional work has also investigated the use of hidden Markov models (HMMs) [4], and, for example, [9] use Gaussian-Processes for non-linear motion prediction, learning the dynamics by employing expectation-maximization and Markov-chain Monte Carlo. More recent work, focuses on more data-driven approaches, with the advancements of Deep Learning. RNNs have become a widely used method for human motion prediction [7] [10] [11], where [1] proposes a 3-layer long short-term memory (LSTM) network, with an encoder-recurrent-decoder model that jointly learns the pose and the temporal dynamics with curriculum learning. The most recent work uses attention-based neural networks along with RNN-based prediction networks, as well as graph convolutional network (GCN) [5].

3 METHOD

3.1 Preprocessing

The main preprocessing step we are doing to our data is calculating its DCT coefficients. This is done in [6] and in [5] to go from sequential data to frequency features. Not only does it allow us to not have to use a deep learning-based solution for temporal encoding, but it also gives us the ability to remove higher frequency motion to make the prediction less jittery. This latter feature is not something that we exploited because our predictions did not seem too jittery.

3.2 Prediction network

For our prediction network, we are using a Graph Convolutional Network (GCN) [3]. In general, graph neural networks (GNN) [8] are an effective way to learn patterns from data that can be represented as graphs. In machine learning literature, we find that GNNs can accurately perform tasks, such as predicting quantum properties of molecules [2], and finding control flow irregularities in code (using the underlying computation graph) to locate logical bugs in software [8]. In our case, they were an intuitive choice, as our task includes working with a joint-based representation of the human body, which resembles a graph - taking the joints as the nodes and their connections as the edges. Using GNNs allowed us not to change the data representation but instead exploit this inherent structure to get more accurate predictions.

In a GNN, each node of the network has an input vector, which represents the features of the node. Additionally, link types can also have numerical representations, which can be taken into account when calculating the output of the network. At each training step t, output vector (embedding), h_t^n, of each node, indexed with n, is updated by applying a function, f, with shared weights, W, to all the neighbouring node's embeddings at timestep $t-1$ (h_{t-1}^n), link type (k), and optionally the embeddings of the node itself ($h_{t-1}^{n_j}$) (where j is the index of the neighbouring node). These values, calculated for all the neighboring nodes, are then aggregated using a chosen method (summing, max-pooling or similar), added to the embedding at the previous timestep (h_{t-1}^n), and a non-linear function, g, is applied to it to get the output embedding for the current step. The overall equation for a generic GNN can be found below.

$$h_t^n = g(h_{t-1}^n + AGGREGATE_{n \to nj}(f(h_{t-1}^n, k, h_{t-1}^{n_j})))$$

During the first step of the training, each node gets information from its neighboring nodes. At the second step, they gain information from all the nodes at a distance of 2. Iterating this way for an appropriate amount of steps information about each node will travel between all node pairs, which enables graph-wide learning of data rules and patterns.

In addition to GNNs, we also took advantage of the concept of graph convolutions. GCNs are a slight modification of the original GNN described above and are empirically shown to improve the

performance of the network. Graph convolutions are similar to convolutional layers in a Convolutional Neural Network (CNN) in that they take into account the spatial structure of the network, or in this case, the connectivity of the nodes. In a GCN, the adjacency matrix is used for this purpose: it's multiplied by the embedding of the previous timestep, H^n_{t-1}, and weight, W, to get the embedding for the next timestep. A very simple example of this equation can be seen below, where A^* is the degree-normalized adjacency matrix, H^{t-1} is the matrix of features of the nodes, and W^{t-1} is the matrix of weights at timestep $t-1$.

$$H_t = \sigma(W_{t-1}H_{t-1}A^*)$$

We also experimented with adding an attention mechanism to the GCN. This is described in detail in the section below.

3.3 Attention network

We have tried out 2 different attention networks to add on top of a GCN. The general schema for an attention network in our case looks like this:

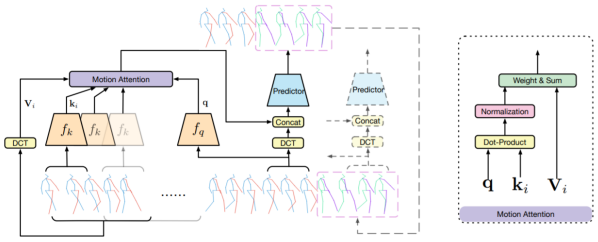

Figure 1: Schema of an Attention network

This figure 1 is taken from [5]. We first tried to use the same attention network that is used in [5]. In this architecture, we use some sort of encoding network to compare the last 17 poses of the whole sequence, which we call the "query", with all the other sequences of length 17 in the whole sequence, which we call "keys". To these keys, we assign a "value", which is a sequence of poses that includes the key and the 24 poses that come after the key. We will compute similarity values between the query and every key using the attention network and then use the weighted sum of the DCT coefficients of the "values" as input to the GCN. We also use the DCT coefficients of the query as input to the GCN, which we concatenate with the weighted sum.

The attention network we are using for our final submission is a two-layer fully convolutional network that generates a feature vector of a sequence of human poses. It computes 1D convolutions across time, using every joint parameter as an input channel. This means there are 135 input channels, and we used 256 different kernels. For a single sequence of human poses, we select the kernel size so that it outputs a vector of length 256. We run the query and the keys through this network and compute the dot product of the outputs. We thus have a similarity value for every key, which we then normalize so that they sum up to 1 and use them for the weighted sum. More information on our novel approach using a GCN encoder/decoder network can be found in 5

3.4 Training

To train this network, we are using the AdamW optimizer, which is known for its stability and performance and reduces the variance seen in network performance due to the random initialization of weights. The learning rate is 0.0005 at the beginning and halves every 400 epochs. We trained our models for 2000 epochs.

4 EVALUATION

To evaluate the different models that we used, we run several experiments, the results related to the training set are summarized in the Figures 2 and Figure 3, and the results related to the validation set, in Figure 4 and 5.

In particular, we show the losses in the vertical axis and the number of update steps in the horizontal axis, and we compare the three best performing models, respectively: the GCN with the encoder/decoder structure in blue, the GCN with the 1D convolution in orange, and the GCN without the attention network in green.

For the validation set, Figure 4 refers to the first study we carried out, analyzing the mean joint angle difference over time, on the validation part of the dataset. The training lasts 500,000 steps and 2,000 epochs, and the two models result with the lowest loss are the GCN network without the attention addition and the GCN network with the 1D convolution. Both obtain a score of 2.52 in the validation set. The other network, GCN with encoder/decoder, obtain a higher loss: 2.56. Similar behavior is observed in the following plot of Figure 5. Here, instead of the mean joint angle difference, the total loss is analyzed, and after 500,000 steps, the 3 models rank in the same way of the previous Figure 5.

In Figures 2 and 3 we show instead the results on the training set. Here, the best performing model is again the GCN with no attention network, with a loss of 2.26 in considering the mean joint angle difference over time, and with a loss of 3.7 for the total loss.

The two best performing models, the GCN with 1D convolution and the GCN without the attention network, obtained in almost all the experiments very similar results.

The results on the public test dataset are interesting: the GCN without any attention network has a mean joint difference of 1.90, the one with the GCN encoder/decoder attention network had a score of 1.88 and the GCN with the convolutional attention network a score of 1.86. This means that while the validation score on the GCN with our novel approach is not as good as just the GCN, it shows potential. Overall, the GCN with the convolutional attention network had the best score on the validation data and on the public test data so we chose this one for our final submission.

Figure 2: Mean joint angle difference over time of the training dataset

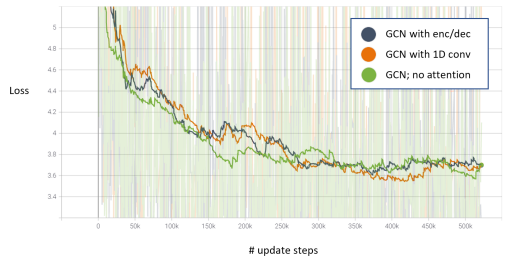

Figure 3: Total loss over time on the training dataset

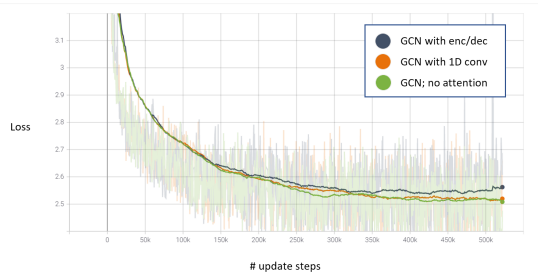

Figure 4: Mean joint angle difference over time on the validation dataset

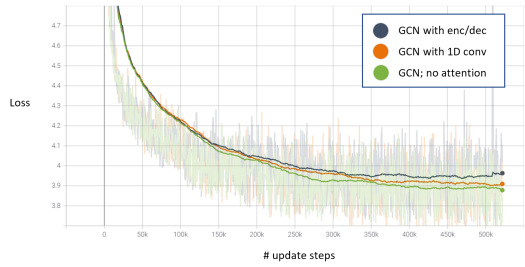

Figure 5: Total loss over time on the validation dataset

5 DISCUSSION

Our first instinct working on sequential data was to use some sort of RNN (Recurrent Neural Network), specifically an LSTM. Since the dummy model which simply flattened the whole time series and used one linear layer achieved a mean joint angle difference of around 7 on the validation set, we thought the LSTM would for surely beat it. However, a reoccurring problem with RNNs and human motion is that they simply learned to output the mean pose of the seed sequence and we were never able to achieve meaningful learning using RNNs.

We have also tried doing some data augmentation, mainly by randomly flipping the time axis in the data. This did not pan out well as human motion data has a direction through time. We thought it would maybe make the model more robust, but it did not improve the validation mean joint angle. We have also tried to add Gaussian noise to the seed sequence of the motion, hoping this would help the model output less jittery data. However, adding Gaussian noise made some of the joint angle matrices invalid for their transformation.

Most importantly, we tried to create a new attention model. We saw that the GCN by itself was great at making motion predictions, so we thought we should try to make an autoencoder using a GCN and using its latent variable representation for computing the values for the weighted sum. We thus created a 2 layer encoder and a 2 layer decoder using graph convolutions. The encoder takes in the keys or the query and runs them through the two layers and creates smaller node embeddings than the original size of each node feature. We then calculate the cosine similarity between the node embeddings of the query and the keys. We thus have a similarity index for each node between the query and the keys. To obtain our attention values, we take the average of those values. Afterward, the calculation is the same as in [5], we calculate the weighted average of the "values" and use this as input for the predictor GCN. To train this network, we ran the keys and query through the encoder and the decoder and summed the l2 loss over each node embedding. As seen in section 4, this network was still able to produce good results, just not as good as the Convolutional Attention network or just the GCN by itself on the validation data. On the public test data, it performed worse than the GCN with the convolutional attention network, but better than the GCN by itself.

Looking at the mean joint angle difference on the validation set, we can see that this attention model started to overfit a little bit. It could be that the Graph Autoencoder starts to overfit the training data since we are training it at the same time as the GCN used for predictions. Using a different type of autoencoder, like a variational autoencoder, could be an area of interest to improve our model and maybe go beyond the performance of the convolutional attention model.

6 CONCLUSION

During this project, we have used the models detailed in [6] and [5] to explore the use of Graph convolution networks for human motion prediction along with the improvement an attention network can make. We have explored the use of a different attention network that uses a GCN autoencoder which showed promising results, albeit not as good as the model in [5]. However, there is a lot of room for improvement and exploration, for example, complexifying the autoencoder or using a variational autoencoder.

REFERENCES

[1] Katerina Fragkiadaki, Sergey Levine, Panna Felsen, and Jitendra Malik. 2015. Recurrent Network Models for Human Dynamics. arXiv:cs.CV/1508.00271
[2] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. 2017. Neural Message Passing for Quantum Chemistry. https://arxiv.org/abs/1704.01212
[3] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. https://arxiv.org/abs/1609.02907
[4] Andreas M. Lehrmann, Peter V. Gehler, and Sebastian Nowozin. 2014. Efficient Nonlinear Markov Models for Human Motion. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 1314–1321. https://doi.org/10.1109/CVPR.2014.171
[5] Wei Mao, Miaomiao Liu, and Mathieu Salzmann. 2020. History Repeats Itself: Human Motion Prediction via Motion Attention. *CoRR* abs/2007.11755 (2020). arXiv:2007.11755 https://arxiv.org/abs/2007.11755
[6] Wei Mao, Miaomiao Liu, Mathieu Salzmann, and Hongdong Li. 2020. Learning Trajectory Dependencies for Human Motion Prediction. arXiv:cs.CV/1908.05436

[7] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1986. Learning representations by back-propagating errors. *nature* 323, 6088 (1986), 533–536.
[8] F. Scarselli, M. Gori, Ah Chung Tsoi, M. Hagenbuchner, and G. Monfardini. 2009. The Graph Neural Network Model. *IEEE Transactions on Neural Networks* 20, 1 (2009), 61–80. https://doi.org/10.1109/tnn.2008.2005605
[9] Jack Wang, Aaron Hertzmann, and David J Fleet. 2006. Gaussian Process Dynamical Models. In *Advances in Neural Information Processing Systems*, Y. Weiss, B. Schölkopf, and J. Platt (Eds.), Vol. 18. MIT Press. https://proceedings.neurips.cc/paper/2005/file/ccd45007df44dd0f12098f486e7e8a0f-Paper.pdf
[10] P.J. Werbos. 1990. Backpropagation through time: what it does and how to do it. *Proc. IEEE* 78, 10 (1990), 1550–1560. https://doi.org/10.1109/5.58337
[11] Ronald J. Williams and David Zipser. 1989. A Learning Algorithm for Continually Running Fully Recurrent Neural Networks.