Pedro Miguel Cera Ramos dos Santos

Skeleton Based Human Activity Performance Evaluation in Tele-Rehabilitation

Dissertação no âmbito do Mestrado em Engenharia Eletrotécnica e de Computadores, Especialização em Computadores orientada pelo Professor Doutor Paulo José Monteiro Peixoto e apresentada ao Departamento de Engenharia Eletrotécnica e de Computadores da Faculdade de Ciências e Tecnologias da Universidade de Coimbra.

Julho de 2023

· U C ·

FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Skeleton Based Activity Performance Evaluation in Tele-Rehabilitation

Pedro Miguel Cera Ramos dos Santos

Coimbra, July 2023

· U C ·

Skeleton Based Activity Performance Evaluation in Tele-Rehabilitation

Supervisor: Professor Dr. Paulo José Monteiro Peixoto

Co-Supervisor: Professor Dr. João Luís Ruivo Carvalho Paulo

Jury:

Prof. Dr. Paulo José Monteiro Peixoto

Prof. Dr. Paulo Jorge Carvalho Menezes

Prof. Dr. António Paulo Mendes Breda Dias Coimbra

Dissertation submitted in partial fulfillment for the degree of Master of Science in Electrical and Computer Engineering.

Coimbra, July 2023

Acknowledgements

The elaboration of this Master's Thesis consisted of a long journey that included numerous challenges and obstacles along the way, which were only overcome with the support and contribution of several people indispensable to the realization of this dream.

Therefore, I want to express my utmost gratitude and appreciation to all those who, directly or indirectly, allowed and contributed to the completion of this Dissertation, as well as to my academic and personal development.

Firstly, I would like to express my gratitude to my advisor, Professor Dr. Paulo José Monteiro Peixoto, from the Department of Electrical Engineering and Computers at the University of Coimbra, for all the guidance, encouragement, availability, knowledge imparted, and for all the ideas and suggestions proposed to bring this project to a successful outcome. Thank you, Professor, for the opportunity to carry out this thesis and for fostering in me the pursuit of knowledge and a critical mindset.

I also want to express my gratitude, in equal measure, to Dr. João Luís Ruivo Carvalho Paulo for his availability and daily guidance demonstrated in the laboratory work, his incredible patience, and the encouragement provided, which never allowed discouragement to set in. I am grateful for all the knowledge imparted during these months.

To the ISR (Institute of Systems and Robotics of the Department of Electrical Engineering and Computers of the University of Coimbra) and the entire team that is part of this institute, I would like to express my gratitude for the welcoming and support provided throughout the completion of this project.

To the entire Alpha Soure 2023 team, from the parish of Soure, especially Table 4, I want to express my gratitude for coming into my life during this stage. Thank you all for bringing more balance and lightness to my life and for becoming family to me.

To my family, I am grateful for all the support and kindness they have always shown me, my mother and my two aunts Maria Teresa and Cristina whom unconditionally supported me in the toughest times. Although all the gratitude in the world is not enough to thank them for everything they have done for me, I especially want to thank my grandparents for their support, encouragement, patience, motivation, dedication, and for all the sacrifices they have made for me to reach this point. I will be forever grateful to you!

To my esteemed friends, those, be they far or near, and those from childhood who helped me with these hurdles. A special thanks to my colleague and close friend Miguel Ângelo Soares Cerveira Varandas, who helped me with this thesis and provided a constant source of laughter.

Lastly, I need to thank my girlfriend Catarina, who for the last 9 years gave me a new focus in life. For all the support she gave me, the laughs and the tears, for walking besides me however steep and ragged the road may be. Thank you for helping me to find a purpose in life.

Resumo

O ramo de aprendizagem computacional, ou *machine learning*, tem sido alvo de bastantes avanços e investimentos tanto do ponto de vista académico bem como em prol da indústria e serviços. Desenvolvimentos e investigação de novas ideias e definições nesta área têm sido uma ocorrência semanal. Estes progressos surgem maioritariamente pelo investimento em aprendizagem profunda (*deep learning*). Os modelos de aprendizagem profunda, alicerçam-se nas arquiteturas de redes neuronais artificiais, que por si, são modelos computacionais inspirados pela estrutura e função do cérebro humano. Estes modelos consistem de várias camadas de neurónios artificiais que são "capazes" de aprender automaticamente e extrair representações hierárquicas dos dados de entrada. Com isto, esta tese propõe-se a implementar um sistema, para a avaliação do movimento esqueleto humano para análise da correção da execução de exercícios de prevenção de comorbidades musculo-esqueletais com base em sistemas de aprendizagem computacional. Este sistema avalia as capacidades do estado da arte na avaliação de um esqueleto virtual. Para esse fim o método separa-se em duas tarefas de avaliação diferentes, sabendo à 'priori' que o movimento tende a repetir-se e a ser uma sequência temporal.

O primeiro é uma rede neuronal recorrente bidirecional (LSTM) com mecanismos de auto-atenção (*self-attention*) de modo a conseguir distinguir entre um exercício bem feito de outro mal feito. O segundo é uma proposta mais ousada, uma rede que permite a predição do movimento tendo em consideração uma sequência contextual. Este método, baseia-se no simples facto de que um esqueleto humano e as suas ligações entre articulações são muito bem codificadas por grafos. Logo a rede a utilizar é uma versão que tem em conta as mudanças espaço-temporais de um grafo, que permite a avaliação da evolução temporal e da interação destas com as juntas. Assim opta-se pela utilização de Redes de Neuronais de Convolução de Grafos, visto que esta, garante a extração de características, *features*, e outras representações dos grafos, conferindo assim, a análise de uma ampla gama de grafos e tarefas.

Vários *datasets* de movimento humano são utilizados, entre eles, o conhecido Human3.6M [1] em formato 3D Cartesiano e 3D Euler, AMASS [2] em formato 3D Cartesiano e por fim um *dataset* não aberto ao público que advém de outro projeto "The PROZIS Challenge". O primeiro método, para reconhecimento de ações, é um dos melhores da literatura, apresenta uma *accuracy* de 96% na classificação da correção da execução de exercícios. Os métodos de predição do movimento são capazes de criar predições fidedignas quando apresentados com um bom contexto, em ações lineares com repetição, como andar ou correr. Estes conseguem prever até 1 segundo ou 30 *frames* a 30Hz sem uma grande acumulação de erro.

Por fim estes métodos são sujeitos a uma bateria de testes e comparados com os demais da literatura, apresentam resultados bastante promissores e com *feedback* visual ilustrador dos erros e da correção do movimento.

Keywords: Machine Learning, Deep Learning, Skeletal Landmarks, Action Recognition, Motion Prediction, Sequence Analysis, Recurrent Neural Networks, Graph Neural Networks.

Abstract

The field of machine learning, has been the subject of significant advances and investments, both from an academic standpoint and in support of industry and services. Developments and research of new ideas and definitions in this area have been a weekly occurrence. These advancements largely arise from investments in deep learning. Deep learning models are based on artificial neural network architectures, which themselves are computational models inspired by the structure and function of the human brain. These models consist of multiple layers of artificial neurons that are "capable" of automatically learning and extracting hierarchical representations from input data.

With this in mind, this thesis aims to implement a system for evaluating human skeleton movement for prevention exercises of various muscle-skeletal comorbidities based on computational learning systems. This system assesses the state-of-the-art capabilities in evaluating a virtual skeleton. For this purpose, the proposed method is divided into two different evaluation tasks, knowing a priori that the movement tends to repeat and be a temporal sequence. The first method is a bidirectional recurrent neural network with self-attention mechanisms to distinguish between well-executed and poorly executed exercises. The second method is a more ambitious proposal: a network that allows movement prediction considering a contextual sequence. This method is based on the simple fact that a human skeleton and its joint connections are well-encoded by graphs. Therefore, the network used is a version that takes into account the spatio-temporal changes of a graph, enabling the evaluation of temporal evolution and the interaction of joints. Thus, the use of Graph Convolutional Neural Networks is chosen, as they ensure the extraction of features and other graph representations, allowing the analysis of a wide range of graphs and tasks.

Various human motion datasets are used, including the well-known Human3.6M dataset [1] in Cartesian 3D and Euler 3D formats, AMASS dataset [2] in Cartesian 3D format, and finally, a non-public dataset provided by the work made by another project, "The PROZIS

Challenge". The first method, for action recognition, is one of the best in the literature, achieving an accuracy of 96% in classifying exercise execution correctness. The motion prediction methods are capable of generating reliable predictions when presented with good context, particularly in linear repetitive actions such as walking or running. They can predict up to 1 second or 30 frames at 30Hz without significant error accumulation.

Finally, these methods undergo a battery of tests and are compared with others in the literature, presenting very promising results and providing illustrative visual feedback on errors and motion correction.

Keywords: Machine Learning, Deep Learning, Skeletal Landmarks, Action Recognition, Motion Prediction, Sequence Analysis, Recurrent Neural Networks, Graph Neural Networks.

"We know it will be hard, we expect it to be long..."

— Sir Winston, Churchill

Contents

Acknowledgements **ii**

Resumo **iv**

Abstract **vi**

List of Acronyms **xiv**

List of Figures **xvii**

List of Tables **xix**

1 Introduction **1**

1.1 Motivation and Context 1

1.2 Problem Formulation 2

1.3 Objective 2

1.3.1 A Note on Machine Learning 3

1.4 Human to Machine Locomotion Representation 5

1.4.1 Joint Representation 5

1.4.2 Pose Modelling 5

1.4.3 Temporal Variability 6

1.5 System Goals 8

1.6 Document Outline 8

2 Background **9**

2.1 Machine learning 9

2.1.1 Brief History of Machine Learning 9

2.2 Recurrent Neural Networks 12

2.2.1 The Recurrent Neural Network Architecture 12

2.2.2 Long-Short-Term Memory Networks 19
2.2.3 Attention Mechanisms 22
2.2.4 Common Attention Mechanisms 24
2.3 Graph Neural Networks 28
2.3.1 Architecture and Pipeline of the Typical GNN 28
2.3.2 The Convolutional GNN 31

3 Related Work **35**
3.1 Human Motion Models 35
3.1.1 RNN Based Classification 35
3.1.2 Recent improvements 37
3.1.3 RNN Based Prediction 38
3.1.4 Graph Based Motion Prediction 41
3.1.5 Considerations on the Transformer Architecture 42
3.2 Skeleton Parameterization 44
3.2.1 Exponential Mapping and Lie Algebra 44

4 Developed Work **47**
4.1 Pipeline of the Proposed Approach 47
4.1.1 Experimental Datasets 48
4.2 Action Recognition 48
4.2.1 Coskun's Self Attentive LSTM 49
4.3 Motion Prediction 53
4.3.1 Space-Time-Separable Graph Convolutional Network 53
4.3.2 Spatio-Temporal Separable Convolutions and Self-Attention 55
4.3.3 Spatial-Temporal Anchor-based Sampling 56
4.3.4 Training 62

5 Results and Discussion **65**
5.1 Metrics and Evaluation 65
5.2 Evaluation of the Baseline Methods 67
5.2.1 Evaluation of Coskun's Attentive LSTM 67
5.2.2 Evaluation of STARS and STS-GCN 69
5.3 Results 72
5.3.1 The PROZIS Challenge Dataset 72

5.3.2 Action Recognition 72
5.3.3 Motion Prediction and Feedback 73

6 Conclusion and Future Work 79
6.1 Future Work 80

A Lie Group Theory and Exponential Maps 98

List of Acronyms

AI	Artificial Inteligence
ANN	Artificial Neural Network
AMASS	Archive of Motion Capture As Surface Shapes
BiLSTM	Bidirectional Long-Short Term Memory
BPTT	Backwards Propagation Through Time
CNN	Convolutional Neral Network
ConvGNN	Convolutional Neural Network
DCT	Discrete Cosine Transform
DGN	Deep Generative Models
DL	Deep Learning
DTW	Dynamic Time Warping
EM	Euclidean Matching
FN	False Negative
FP	False Positive
FPR	False Positive Rate
GAT	Graph Attention Network
GCN	Graph Convolutional Network
GNN	Graph Neural Network

IESTGCN	Interaction-Enhanced Spatial-Temporal Graph Convolutional Network
IID	Independent and Identically Distributed random variables
INPACT	Intelligent Platform for Autonomous Collaborative Telerehabilitation
ISR	Institute of Systems and Robotics
JPE	Join Position Error
LNLSTM	Layer Normalized Long-Short Term Memory
LSTM	Long-Short Term Memory
ML	Machine Learning
MLP	Multi-Layered Perceptron
MMD	Maximum Mean Discrepancy
MPJPE	Mean Per Joint Position Error
NCA	Neighbourhood COmponents Analysis
NMI	Normalised Mutual Information
R and D	Research and Development
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Networks
ROC	Receiver Operating Characteristic
SGD	Stochastic Gradient Descent
SRNN	Structural Recurrent Neural Network
ST	Spacio-Temporal
STARS	Spatial-Temporal Anchor-based Samplin
STSGCN	Space-Time Separable Graph Convolutional Network

TN	True Negative
TP	True Positive
TPR	True Positive Rate

List of Figures

1.1 Spine analysis system's user feedback [16] 3

1.2 Virtual joint representation of the human body. 6

1.3 Differences Between Euclidean Matching and Dynamic Time Warping [38]. . 7

1.4 Simple RNN and its unfolding through time, writing as a network for the complete sequence [39] 7

2.1 Range of Simple RNNs 12

2.2 Similar to 1.4, this Figure shows the unfolded state of a RNN layer and the connections between recurrent units [54]. 13

2.3 Common types of activation functions. 15

2.4 Differentiable Function [57]. 16

2.5 Non-Differentiable Function [57]. 16

2.6 Convex and Non-Convex Function [57]. 16

2.7 LSTM Cell [64] 20

2.8 Unrolled Bidirectional LSTM [67] 22

2.9 The attention mechanism computes a linear combination over values v_i via attention pooling, where weights are derived according to the compatibility between q and k_i [64] 23

2.10 Multi-head attention involves concatenating multiple heads and subsequently linearly transforming them [64]. 26

2.11 Comparison of the CNN, RNN and Self Attention architectures [64] 27

2.12 Conversion of nodes to feature vectors in a GNN [78] 29

2.13 Architecture of a GNN [78] 30

3.1 Measurement of the similarity to a query sequence (Walking) as per [88] . . 35

3.2 Motion prediction [24].
. . . . 35

3.3 LSTM3LR architecture 39
3.4 ERD based on the LSTM3LR architecture. 39
3.5 Fragkiadaki et al.'s work in RNNs for motion recognition [23], [105] 39
3.6 Martinez et al. implementation of short time prediction [23] 40
3.7 Learnt joint relations following a kinematic tree and other connections between joints [25] 42
3.8 An example spatio-temporal graph in the human activity context [26] 42

4.1 High level pipeline of the proposed method. 48
4.2 Coskun et al.'s Network. [88] a) the architecture itself. and b) the attention based model that uses layered normalization 50
4.3 Pipeline of the STS-GCN according to Sofianos et al. [25] 53
4.4 Spatial-Temporal AnchoR-based Sampling [102] 57
4.5 The optimization process involves the joint optimization of anchors and network parameters [102] 58
4.6 Combination of ST anchors, sampled noise with the backing STGCN [102] . 60

5.1 The F1 scores and Normalized Mutual Information of 3 of the best baselines and Coskun et al.'s method MMD-NCA [88] 68
5.2 Attention mechanism in play [88] 68
5.3 Results for an experiment in the conditions above stated in 4.2.1 75
5.4 Sequence of 25 images in a walking exercise 76
5.5 Sequence of 25 walking frames proving the importance of historic context . . 78

List of Tables

2.1 Notable Advancements in AI and ML, by [48] 11
2.2 Comparison of Complexity, Sequential Operations and Maximum Path Length as per Vaswani et al. and Aston Zhang et al. [64], [70] 27

5.1 Comparison of the FPR 67
5.2 Comparative analysis of the obtained results. 70
5.3 Average Comparison between multiple methods. 71

1 Introduction

Advances in technology and computing capabilities have led to the search for digital solutions to address inherent human ills. Usually, the digitalization of medicine refers to the transformation of analogue storage systems, i.e. paper-based, networked electronic patient record systems [3]. However, this view of digitalization is incomplete.

In general, digitalization is and has become a very important innovation in all areas of daily life [4], [5]. The greatest impact of digitalization in the 21st century has focused on the influence of digital innovations in medicine and physical well-being [4], [6]. Some of these innovations will have minimal impact on society, while others will have a huge impact [4], [7] that can be considered technological *game changers*.

1.1 Motivation and Context

Medical care has developed at an unprecedented pace. The large amount of information and data, as well as their reliability, lead to a change in the interaction between the patient and the medical professional [4]. Physiotherapy, as one of the areas of healthcare, can be improved through digitalisation, which allows for better collaboration between the two parties mentioned above, as well as a significant improvement in the quality of life of patients [8]–[11].

As a rule, participation in physiotherapy and rehabilitation measures is essential [12] for post-operative recovery or even as treatment for various musculoskeletal diseases [12]. However, the presence of a professional for all rehabilitation interventions may sometimes not be feasible or economically justifiable [12], [13]. Thus, current health care systems (despite some steps taken during and after the Covid 19 pandemic) continue to be based on rehabilitation programmes in which a first phase of treatment is provided in person in special facilities under the direct supervision of a health professional. In the second phase, the patient is advised to perform rehabilitation exercises in an external environment, usually at home [14].

1.2 Problem Formulation

The incidence of musculoskeletal problems is increasing due to an inactive lifestyle and an ageing population. As a result, the demand for rehabilitation services to maintain people's mobility and functionality is increasing. However, the provision of continuous care is a challenge for health systems and places a social, economic and psychological burden on patients and their families. Telerehabilitation is a potential solution that allows individuals to take control of their condition and improve their overall well-being. This approach offers significant cost savings by eliminating travel expenses, in-person therapy sessions, absenteeism and medication. It also allows for more frequent training sessions from the comfort of home, promotes motivation through personalised programmes and incorporates interactive elements such as serious games to increase treatment adherence. The effectiveness of telerehabilitation is scientifically proven and shows comparable or even better results than conventional interventions [15].

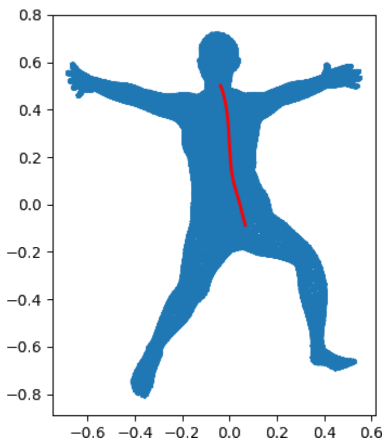
1.3 Objective

Due to the periodic nature of movement and exercises, some assumptions can be made. The exercises tend to be repetitive, of unknown lengths and data quantity.

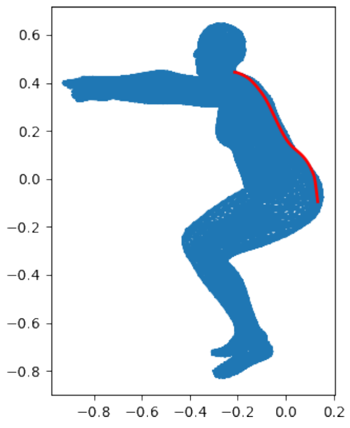
This project is part of a bigger project (INPACT) of the Institute of Systems and Robotics (ISR), which aims to create a cost-effective and to facilitate access to rehabilitation services and ensure greater equity within a portable station, which also has an impact on shortening waiting lists. It allows for the development of a telerehabilitation platform with a user interface that suggests exercises pre-configured remotely by a therapist. The proposed system monitors the user's performance and provides real-time feedback. It is innovative in that it provides holistic visual perception of the user's body movements, without the use of markers and with an autonomous ability to analyse performance using machine learning techniques. The information obtained for each user is centralised in the cloud, allowing autonomous temporal analysis of the user's performance over multiple rehabilitation sessions and the generation of alerts to the therapist whenever there is a deviation from the plan [15]. Due to the need to provide feedback internally and externally in the context of physiotherapy and rehabilitation, the comprehensive Intelligent Platform For Autonomous Collaborative

Telerehabilitation (INPACT) project was created.

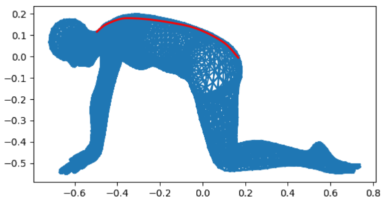
Nevertheless, the INPACT project will also try to improve on the existing state of the art by implementing new and novel machine learning technologies such a motion prediction feature using skeleton based data and a spinal curvature assessment, using a 3D mesh character allowing a precise fit to the back regions, as show in Figure 1.1 [16], [17].



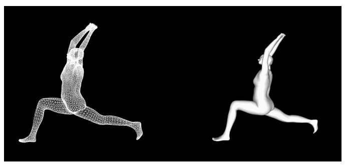
(a)



(b)



(c)



(d)

Figure 1.1: Spine analysis system's user feedback [16]

This work aims to be at the local backend part of the project. More specifically, as the title suggests, evaluating the "goodness" of movements with the help of a virtual skeleton using robust Machine Learning (ML) solutions and outputting a set of parameters that can be used to provide feedback to the user. Thus, this thesis will not focus on the skeleton extraction and front end of the project, even if some general guides will be given throughout the rest of this document. Thus, the input is assumed as being an already extracted skeleton and through the use of ML methods, relevant metrics and feedback will be provided to the user, Such as an overall metric score of the goodness of the execution or even a representation of this movement through time.

1.3.1 A Note on Machine Learning

Pertaining to the previously made assumptions, ML approaches are one of the best choices for human movement and sequence analysis. Thus, compared to other traditional methods, ML has some advantages:

- Automatization: ML algorithms can analyse large volumes of video data automatically, without the need for manual intervention.
- Scalability: ML models can scale to analyse vast amounts of video data.
- Adaptability: ML models can adapt and learn from new data, allowing them to improve their performance over time. As more video data becomes available, machine learning models can continuously update and refine their understanding, leading to better analysis results.
- Pattern recognition: Probably the most important aspect, ML excels at recognizing patterns in video data. Convolutional Neural Networks (CNNs), a popular class of machine learning models for image and video analysis, are designed to detect and classify visual patterns. These models can detect objects, track movement, recognize actions, and even understand complex scenes, surpassing the capabilities of many traditional methods.
- Real-time analysis: ML models can be optimized to perform video analysis in real-time or near real-time. In most cases, the inference process takes less time to execute than its traditional counterpart, possibly allowing for a real time analysis or at least a faster inference process [18].

In general, within the spectrum of artificial neural networks (ANNs), convolutional neural networks (CNNs) and recurrent neural networks (RNNs) have been the best choice within the spectrum of artificial neural networks (ANNs) for image processing and sequence analysis though there are novel architectures and other approaches rising in usage.
CNNs are similar to traditional ANNs in that they consist of neurons that self-optimise by updating the weights along with the extraction layers for the features, through an algorithm called backpropagation. In practise, each neuron receives an input signal needed for operation (scalar non-linear function). By default, CNNs are coupled with image classification models [19].

1.4 Human to Machine Locomotion Representation

Man has perfected the prediction and analysis of the evolution of the world around him over millions of years. How are we able to make short-term predictions about the evolution of various environmental phenomena? Be it catching falling fruit or shooting clay for fun. How do boxers know when to dodge a punch? This short-term prediction is an amazing aspect of human physiology.

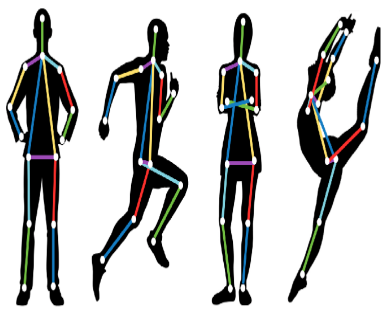
For these and many other reasons pertaining to this thesis, a machine must adapt its behaviour and focus its attention on certain parts when interacting with humans [20]. Therefore, digital human pose representation is essential, as shown in the Figure 1.2, skeleton-based pose representations are inherently attractive due to their comparatively small number of variables, as they are robust to changes in viewpoint, scale, movement speed and real-time performance [20]–[22].

1.4.1 Joint Representation

Most systems for parameterising the human body for a digital skeleton use one of four forms: Cartesian 2D or 2D joint angles, Cartesian 3D coordinates and 3D angles. These are usually calculated based on an RGB image. Then, using computer vision algorithms or neural networks, these skeletal key-points, or joints, are extracted as a 2D model or as a 3D estimation of the 2D model. However, Julieta Martinez et al. [23] applies a transformation into residuals where velocity can be useful, using frequency to help with periodic motion. Wei Mao et al. [24] also use a discrete cosine transform (DCT), which is excellent for analysing periodic motion [25]. Ashesh Jain et al. [26] and Zhenguang Liu et al. [20] used novel approaches using Lie algebra parametrisations in exponential maps.

1.4.2 Pose Modelling

To represent poses and describe human movement, more orthodox methods follow mathematical models of representation that fall into one of two categories: Top-down approaches, which introduce latent states to describe the temporal dynamics of movements, and bottom-up approaches, which use local features to represent movements [12]. Traditional methods in the first category include Kalman filters, hidden Markov models or even Gaussian mixed models [12], [20], [29]–[31]. These methods are not the "be-all and end-all" of motion representation and have some drawbacks, in the case of Kalman to linearise the


(a) Keypoint capture in a human silhouette [27].

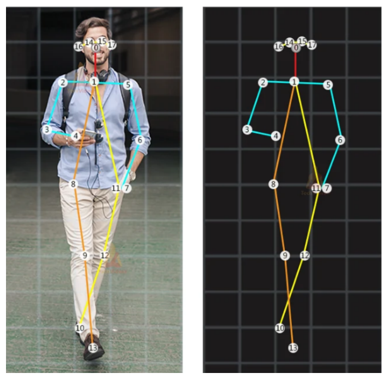

(b) Skeleton keypoint representation of human in a RGB image [28].

Figure 1.2: Virtual joint representation of the human body.

transitions between latent states or the assumption of simple structures of the latent states themselves, i.e. Hidden Markov. Bottom-up approaches rely on the extraction of local features in order to use predefined criteria in identifying key points, postures or statistics derived from these (average, median, standard deviation). This limits the ability to deal with temporal variations within the data [12], [32]–[34].

1.4.3 Temporal Variability

Time series prediction is a very important area of research because many types of data change along the time axis. In the classical models of ML, the feature engineering is done manually, and the parameters are optimised considering the domain model [35]. Therefore, there are some clever matching techniques between temporal sequences, i.e. Euclidean Matching (EM) and Dynamic Time Warping (DTW), the latter being the more popular of the two due to its point-to-point matching compared to one-to-one matching via EM [36]–[38]. DTW is thus used to check similarities between two temporal sequences with variable length and variable speed, as illustrated in Figure 1.3 [37], [38].

Nevertheless, there have been advances in deep learning and machine learning algorithms that have enabled the processing of time series. As the number of layers increases, the complexity of the Artificial Neural Network also increases, commonly referred to as "deep learning" [39], [40]. In general, the layers in ANNs consist of simple non-linear units. The higher the number of layers, the more abstract the data representation and the suppression of unwanted variability [39], [40].

Recurrent Neural Networks, (RNNs) are ANNs with recurrent connections that are

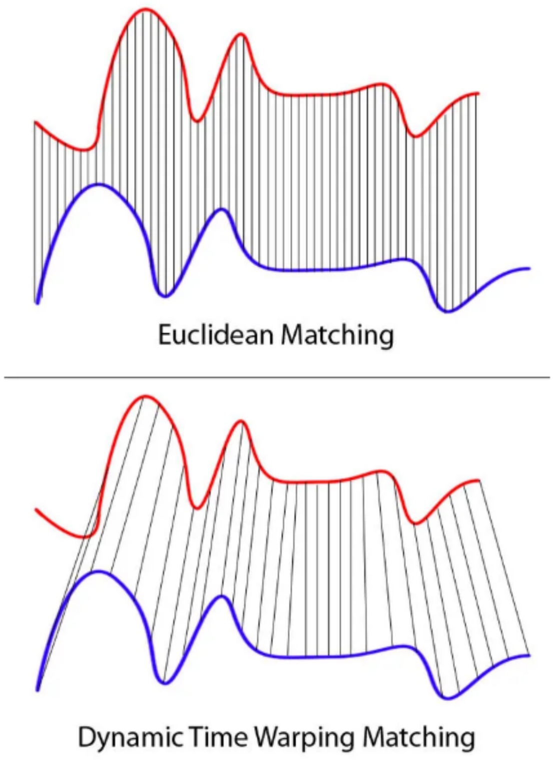


Figure 1.3: Differences Between Euclidean Matching and Dynamic Time Warping [38].

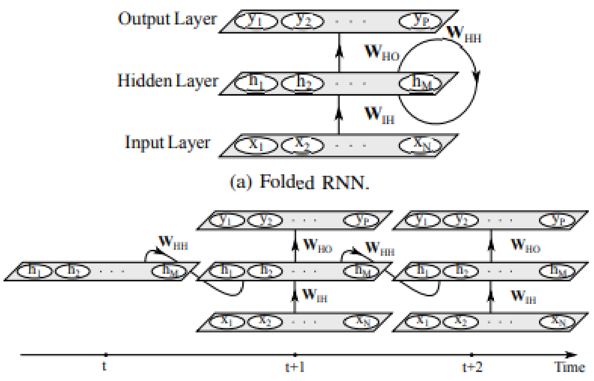


Figure 1.4: Simple RNN and its unfolding through time, writing as a network for the complete sequence [39]

capable of modelling sequential data for sequence recognition or prediction [39], [41]. Structurally (Figure 1.4), the hidden states of a recurrent unit act as the "memory" of the network and the state of the layer is determined by the previous layer [40], [42]. Thus, this type of network can chop and process past signals over time, map an input sequence to the output at the current time step, and predict the sequence at the next time step [39], [41], [42].

1.5 System Goals

As aforementioned, the proposed system will use a ML approach that should be capable of processing a large quantity of skeleton-parameterized sequences, thus some requirements can be defined:

- Analysis of sequences of varying lengths.
- Providing a robust model capable of detecting execution errors.
- Implementation of novel ways to qualify movement accuracy.
- Capable of giving visual feedback and other metrics for the evaluation of movement.

1.6 Document Outline

The document is split in 6 main chaptersI In this first chapter, the requirements are outlined, and the problem formulated, as well as the proposal for the solution found.

In the second chapter, an overview of the machine learning base methods will be given, and how these methods influence the chosen direction of this work's pipeline.

The third chapter consists of a description of the methods of the state-of-the-art, as well as their pros and cons.

The fourth chapter focuses on the detailed inner workings the selected architectures for this thesis as well as giving a needed insight on data parameterization and preprocessing

After this, in the fifth chapter, both the quantitative and qualitative results of the methods will be described as well as the datasets used for the extracted quality metrics.

Lastly, chapter six is used for the presentation of the conclusions drawn from the extracted metrics and other qualitative features, as well as some notes about the future work to enhance or improve the current methods.

2 Background

This chapter will give a theoretical overview of the state-of-the-art architectures used in some common Machine Learning algorithms and the reasoning behind them. Due to the broad spectrum of possible solutions, some approaches were purposely left out, during the early stages of the project.

2.1 Machine learning

ML consists of a branch in the field of artificial intelligence based on the use of statistical models to make predictions. This is usually described as a form of predictive analysis, where the function of the computer is to learn a task without being explicitly programmed to perform it [14]. In general, ML algorithms use empirical data and analyse it based on algorithms for minimising nonlinear error functions [12], [43].

2.1.1 Brief History of Machine Learning

In this half of the twenty-first century, due to many factors, including ever-expanding computational capabilities, ML is a tool to harness technologies around artificial intelligence because of its learning and decision-making capabilities [44]. As Zhou notes [45], ML is a product of AI studies in the late 1940s, 1950s and 1970s, when it was ambiguously thought that machines could become intelligent if they could think logically. According to Hebb [46], machine learning is based on the interaction between brain cells, on neuron excitation and neuron communication [44]. Hebb's model can be described as a way to change the relationship between neurons (nodes) [44]. This relationship can be strengthened when two neurons are activated simultaneously or weakened when they are activated separately, the strength of this so-called relationship is known in the literature as "weight" [19], [44], [47].

In the 1950s, Arthur Samuel of IBM developed an algorithm for playing checkers. This algorithm used a scoring function that attempted to measure the probability of winning and chose its next move using a min-max strategy [44]. In 1957, psychologist Frank Rosenblatt, working at Cornell Aeronautical Laboratory, developed the first perceptron using Hebb's model of brain cell interaction and Arthur Samuel's work. The software developed for the IBM 704 was used for image recognition [44]. In the Table 2.1 there can be found a timeline of some achievements of research in artificial intelligence.

Year	Contributor(s)	Contribution
1873	Alexander Bain	Introduction of Neural Groupings, early models of neural networks
1943	McCulloch and Pitts	MCP Model. The ancestor of ANNs
1949	Donald Hebb	Hebbian learning rule, the foundation of modern NNs
1958	Frank Rosenblatt	Introduction of the first proto-perceptron
1974	Paul Werbos	Introduction of backpropagation
1980	Teuco Kohonen	Introduction of the Self Organizing Map
1980	Kunihiko Fukushima	Introduced Neocogitron (proto-Convolutional NN)
1982	John Hopfield	Introduction of the Hopfield Network
1985	Hilton and Sejnowski	Introduction of the Boltzmann machine
1986	Paul Smolensky	Introduction of Harmonium (Restricted Boltzmann machine)
1986	Michael I. Jordan	Definition and introduction of the Recurrent Neural Network
1990	Yan LeCun	Introduction of LeNet, confirmation of the possibility of deep neural networks
1997	Schuster and Paliwal	Introduced Bidirectional RNNs
1997	Hochreiter and Schmidhuber	Introduction of LSTM
2006	Geoffrey Hinton	Introduction of Deep Belief Networks
2009	Salakhutdinov and Hinton	Introduction of Deep Boltzmann Machines
2012	Geoffrey Hinton	Introduction of Dropout
2013	Graves	Introduction of Stacked LSTMs
2015	Mikolov	Structurally constrained RNN
2017	Jing	Introduction of Gated orthogonal recurrent units

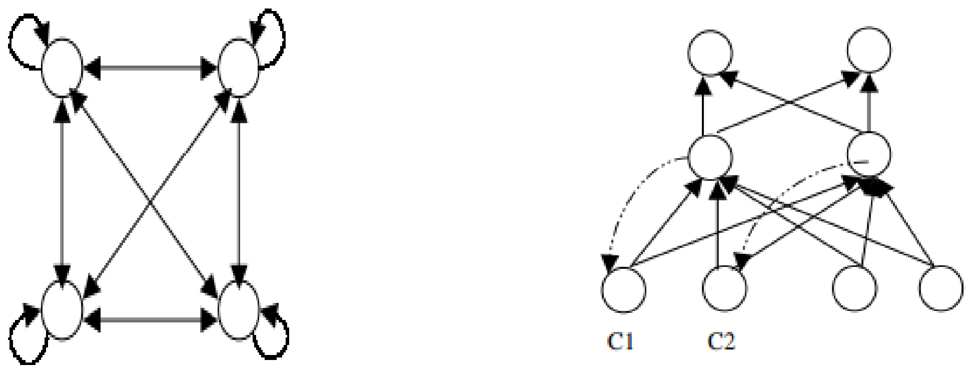
Table 2.1: Notable Advancements in AI and ML, by [48]

2.2 Recurrent Neural Networks

Recurrent Neural Networks have, since the mid-'90s, been the foci of a good deal of research and development teams [49]. RNNs are a part of supervised training models with closed loop (feedback) connections, designed to learn time-varying data and sequences [39], [49]–[51]. RNNs have been applied to a varied list of problems. By the late '80s, some partially recurrent networks were published by many researchers of renown, like Rumelhart, Hinton and Williams to learn strings of characters [49]. Since then, RNNs have been used in countless topics, like object tracking, data forecasting and other time-sensitive minimization tasks.

2.2.1 The Recurrent Neural Network Architecture

Traditional RNN consists of three layers. The input, the recurrent layer and the output layer. The output of the last time step is used as the input of the current time step [39], [52]. The input layer has N inputs, where each input is a sequence of vectors through discrete time, such as $\{..., x_{t-1}x_t, x_{t+1}, ...\}$, whence $x_t = (x_1, x_2, ..., x_N)$ [39], [52], [53]. Thus, according to Jain and Medsker, [49], RNN architectures range from fully connected and partially connected (Figure 2.1), the latter being used to learn strings of characters in the early '80s, hence the focus of this thesis will be on the former. Thus, in a fully connected



(a) Simple fully connected RNN [49] (b) Simple partially connected RNN [49].

Figure 2.1: Range of Simple RNNs

RNN, the input units are connected to the hidden layer, with the connections parameterised by a weight matrix W_U [39], [53], [54]. The connections from the hidden to hidden layer are parameterised by the weight matrix W_W and finally the connections from the hidden to the output layer are mapped by W_V. These weights $W_{U,W,V}$ are shared across different time steps [39], [54].

Hidden layers and Recurrent Units

The hidden layer has M recurrent units $h_t = \{h_1, h_2, ..., h_m\}$ connected by recurrent links. It is important to note that initialising each hidden unit with small but non-zero elements can improve the stability and performance of the network[39], [53], [54]. The space

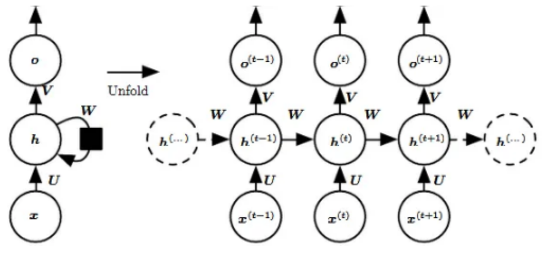


Figure 2.2: Similar to 1.4, this Figure shows the unfolded state of a RNN layer and the connections between recurrent units [54].

state, or "memory" is defined by the hidden layer as the following equations:

$$h_t = f_H(o_t) \tag{2.1}$$

Given that:

$$o_t = W_U\, x_t + W_W\, h_{t-1} + b_h \tag{2.2}$$

$f_H()$ being the activation function of the hidden layer, and b_h the bias vector of the hidden units. Thus, the output layer, having P units where, $y_t = (y_1, y_2, ..., y_P)$, connected to the output layer through weighted connections W_V computed as:

$$\mathbf{y}_t = f_O(W_V\, h_t + b_o) \tag{2.3}$$

Here, $f_O()$ is the activation function of the output layer, assuming the same thought process, b_o being the output layer bias [39], [49], [52]–[54]. Since the pairs input-target are sequential through time, the above equations need to be consequently repeated over finite time steps $t = 1, ..., T$. The general steps to follow are:

1. Initialize the weight matrices $W_{U,W,V}$, following a random distribution bias, with small non-zero values.

2. Forward propagation to calculate predictions.

3. Compute the loss.
4. Backpropagation to calculate the gradients.
5. Update the weight matrices based on the computed gradients.
6. Repeat steps 2-5 through each time step.

The equations 2.1 and 2.3 show that RNNs make use of non-linear, time iterable state equations [39], [49], [52]–[54]. At each time step, a prediction is made and passed between the hidden states and the output layer, as shown in the equations 2.1 and 2.2, h_t is calculated using the current input and the hidden state of the previous time step [54]. These hidden states summarise the unique necessary information of past states across multiple time steps [39]. RNNs use simple non-linear activation functions for all units. This enables the modelling of complex dynamics [39], [49], [54].

Activation Function

Regardless of how many hidden layers a network has, if the activation function is linear, these multiple "linear" hidden layers behave like a single hidden layer [41]. The most commonly used activation functions include the rectified linear unit (ReLU), the sigmoid function and the hyperbolic tangent and some others as shown in Figure 2.3. The sigmoid function, a known activation function, takes a real value and "squeezes" it into the range [0,1] [39], [53], [54]. The sigmoid is usually used in the output layers, where a cross entropy function is used. The hyperbolic tangent (tanh), is a scaled version of the sigmoid [39]. These are defined as:

$$tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{2.4}$$

and

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.5}$$

or

$$\sigma(x) = \frac{tanh(\frac{x}{2}) + 1}{2} \tag{2.6}$$

ReLU is also a very common function, due to its simplicity and fast computing time, even though it is open-ended for positive input values [39]. it is defined as:

$$y(x) = max(x, 0) \tag{2.7}$$

The activation function depends on the type of data, i.e. the sigmoid function is normally used in networks where the output is between 0 and 1. However, the sigmoid and hyperbolic tangent functions saturate very quickly and can cause problems with the vanishing gradient, leading to unstable dynamics in the gradient updates [52], [53], [55]. Therefore, the ReLU activation function can lead to sparser gradients and significantly accelerate the convergence of the stochastic gradient descent [41], [56].

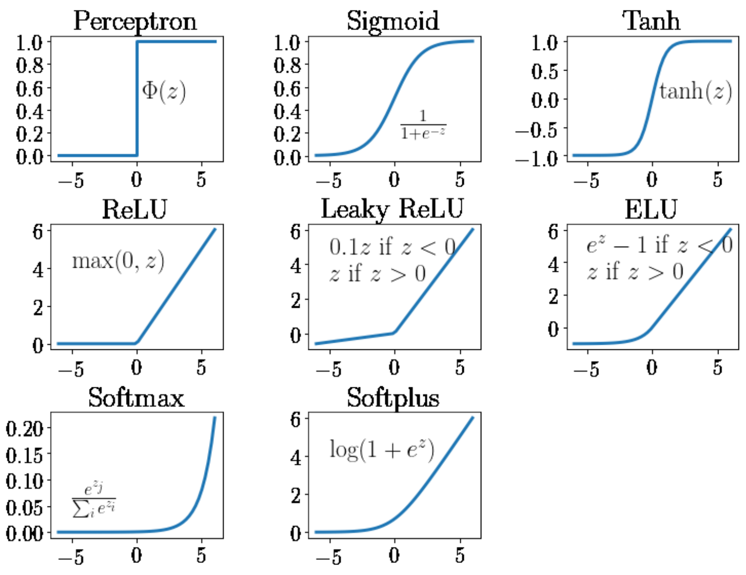


Figure 2.3: Common types of activation functions.

However, ReLU does not perform well when paired against a large gradient flow. As the weight matrix grows, neurons may remain inactive during training [56].

Learning Methods and Gradient Descent

The focus of any neural network is to minimise/maximise a particular function through an optimisation method. Gradient descent is a very popular and simple method to find the local minimum/maximum. In ML and Deep Learning (DL) this method is used to find the local minimum of a given loss function [41], [57]. Most other optimisation methods are based on the concept of gradient descent. Thus, in most gradient descent algorithms, an initial point must be chosen, and the algorithm is performed as follows:

$$w^k = w^{k-1} - \alpha \nabla g(w^{k-1}) \tag{2.8}$$

where α is the step size or more commonly known as the learning rate and w^{k-1}, the current position which paired with the next step, w^k denotes the direction to follow and $g()$, the multivariate differentiable function in a neighbourhood of the point w^{k-1}. This minimization must follow the first-order optimality condition, as shown in equation 2.9 [41], [58]. The first derivatives of a function are always zero at the function's minima. This happens because minima are naturally found at the bottom of valleys where the tangent line or hyperplane touching the function is completely flat, resulting in a slope of zero.

Since the derivative or gradient at a specific point provides information about this slope, the values of the first derivatives serve as a useful means of identifying minimum values of a function $g()$, in the case of N being one at any point w [58].

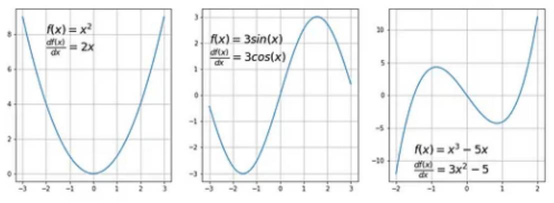


Figure 2.4: Differentiable Function [57].

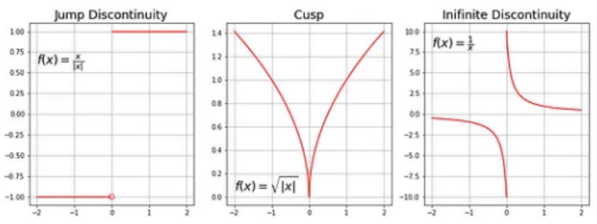


Figure 2.5: Non-Differentiable Function [57].

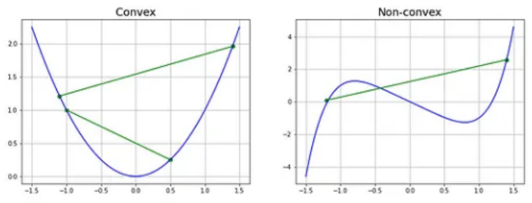


Figure 2.6: Convex and Non-Convex Function [57].

For the gradient descent algorithm to work, the objective function must be differentiable and convex. This means that the function must have a derivative for every point in its entire range and be a univariate function. The line segment connecting two points must lie on or above the curve and must not cross it, some examples are given in Figure 2.6. Meaning that the function to optimise has more than one minimum.

$$\nabla g(w) = 0_{N\,x\,1} \tag{2.9}$$

As mentioned earlier, other common optimisers are improved versions of the basic gradient descent method. One step in improving the optimiser is to add momentum, a method that attempts to slow the rate of descent near the optimum to avoid overshooting. Therefore, some very common functions are based on Nesterov momentum, such as the SGD optimization function [58].

The Nesterov Momentum can be defined as:

$$m^k = \beta m^{k-1} - \alpha \nabla g(w^{k-1}) \tag{2.10}$$

Improving the basic gradient descent by computing the approximate gradient of the next position. Producing faster convergence, normally used in the Stochastic Gradient Descent

(SGD), with Nesterov momentum updates m^{k-1}, for the current step and m^k for the next time step [41], [58], [59]:

$$w^k = w^{k-1} + m^k \tag{2.11}$$

Another common algorithm is the Adagrad (Adaptative Learning Rate). This method does not use momentum, so it is simpler compared to SGD. Adagrad bases itself on the usage of different learning rates for each parameter base on iteration [59]. The reasoning behind this is that different learning rates are needed when dealing with data of different sparsity [58], [59]. Adagrad can be described by the following equations:

$$s^k = s^{k-1} + \nabla g(w^{k-1}) \otimes \nabla g(w^{k-1}) \tag{2.12}$$

$$w^k = w^{k-1} - \alpha \nabla g(w^{k-1}) \oslash \sqrt{s^k} + e \tag{2.13}$$

Adding to the previous notation, s, denotes the sum of the sum of all the previous time step's squared values of the gradient and e a small number to avoid division by zero. As shown in the Adagrad equations, the goal is to reduce the gradient vector along the steepest dimensions. Solving the gradient descent problem, where the steepest slope does not necessarily point to the global optimum [58]. This changes the learning rate as the sum of the previous gradient square increases after each step [58], [59].

A small improvement of Adagrad is RMSprop, which avoids the effect of monotonically decreasing the learning rate by adding the hyperparameter β which is the decay rate [58], thus, RMSprop can be defined:

$$s^k = \beta s^{k-1} + (1-\beta)\nabla g(w^{k-1} \otimes \nabla g(w^{k-1}) \tag{2.14}$$

Another widely used improvement to the gradient descent algorithm is the Adam optimisation function. Adam stands for Adaptative Momentum Estimation and combines the notion of momentum with that of Adagrad, or more precisely an improvement of Adagrad (RMSprop) [58]. Adam is an extension of Adagrad and attempts to solve for decreasing learning rates by using exponentially weighted averages, i.e. instead of calculating the sum of all past squared gradients in t time steps, the window size is constrained and the average of the gradients of said window is calculated [59]. Therefore, the Adam optimiser is described as follows:

$$m^k = \beta m^{k-1} - (1-\beta)\nabla g(w^{k-1}) \tag{2.15}$$

$$s^k = \beta s^{k-1} + (1-\beta)\nabla g(w^{k-1}) \otimes \nabla g(w^{k-1}) \tag{2.16}$$

$$\hat{m}^k = \frac{m^k}{1-\beta_1^{\mathrm{k}}} \tag{2.17}$$

$$\hat{s}^k = \frac{s^k}{1-\beta_2^{\mathrm{k}}} \tag{2.18}$$

$$w^k = w^{k-1} + \alpha\hat{m}^k \oslash \sqrt{\hat{s}^k} + e \tag{2.19}$$

where β_1 is the momentum decay hyperparameter and β_2 the scaling decay. Moreover, m and s are then bias corrected as per equations 2.17 and 2.18. There are even other optimisation methods, such as Newton's method, which takes into account not only the gradient but also the curvature. Since this work focuses on networks based on first-order optimisation algorithms, second-order optimisation functions are not presented.

Loss Function

So what is a loss function? The loss function evaluates the performance of the network by comparing the output $\mathbf{y}_t$ with the desired objective z_t. Some of the loss functions are shown in the below equations [39]:

$$Loss : L(y,z) = \sum_{t-1}^{T} L_t(y_t, z_t) \tag{2.20}$$

$$LeastSquares : L(y,z) = \frac{1}{2}(z_t - y_t)^2 \tag{2.21}$$

$$CrossEntropy : L(y,z) = -y_t log(z_t) \tag{2.22}$$

The total loss function is calculated as the sum of the individual losses at each time step [39], [58], [60]. The choice of the loss function depends on the problem at hand. Commonly used loss functions include Euclidean and Hamming distances for forecasting real values and the cross entropy applied to the probability distribution of outputs for classification tasks [39], [58], [60].

Forward Pass

At each time step during the forward pass, the hidden layer, h^t (as per previous notation), of the network receives an external input and also considers its activation from the previous time step across all dimensions. The combination of the input and the previous activation of the hidden layer is sequentially fed into the network based on the input sequence. As a result, the network stores the resulting hidden layer activation [39], [54], [58]. In this way, the network propagates the current input signal further and allows it to flow through the different layers. The activation of the current context, a^t layer is then used in the process [49].

The forward pass can then be described by the following equations:

$$a_t = b + W_W h_{t-1} + W_U x_t \tag{2.23}$$

$$h_t = f_a(a_t) \tag{2.24}$$

$$o_t = c + W_V h_t \tag{2.25}$$

$$\hat{y}_t = softmax(o_t) \tag{2.26}$$

Thus, for a given sequence of input values x paired with a sequence of target values y, the total loss is calculated by summing the losses over all time steps. In this case, the outputs o^t are passed through the softmax function (if used) to produce a vector $\hat{y}$ representing the probabilities of the output. The loss L is then determined as any of the existing loss functions of the true target y^t given the inputs received [54].

Backward Pass

The process of calculating gradients involves two passes. A forward propagation pass from left to right, followed by a backward propagation pass from right to left (as shown in the Figure 2.2). The runtime for this process is $O(n)$, and cannot be reduced by parallelisation because the forward propagation graph inherently requires sequential computation. Each time step builds on the previous one, so it must be computed in the correct order. The states computed during the forward pass must be stored until they are used again in the backward pass, resulting in a memory cost of $O(n)$. This back-propagation algorithm, with a cost of $O(n)$, is called back-propagation through time (BPTT). Due to the parameters shared across all time steps, the gradient at each output depends not only on the computations of the current time step, but also on the computations of the previous time steps [49], [54].

2.2.2 Long-Short-Term Memory Networks

Recurrent connections in neural networks can improve performance by exploiting their ability to capture sequential dependencies [61], [62]. However, the effectiveness of these connections is often limited by the training algorithms used for RNNs [39], [62]. Many existing models have problems with exploding or vanishing gradients during training, leading to difficulties in learning long-term dependencies in sequential data. Special models have been developed to overcome this challenge, the most popular being long short-term memory (LSTM) RNNs. The LSTM is widely recognised as an efficient solution for mitigating the problems associated with vanishing and exploding gradients [39], [61]. In LSTM, the structure of the hidden units is modified to incorporate memory cells that are controlled by gates. These gates regulate the flow of information to the hidden neurons and hold important extracted features from previous time steps [39], [61], [62]. This thesis will not discuss all variants of LSTM, i.e. S-LSTM or even Stacked and Grid-LSTM. An insight into the inner workings of the standard architecture is given and deepened by the introduction of bidirectional LSTM, which is relevant to this work.

The Standard Architecture of an LSTM

The standard LSTM cell according to Hochreiter & Schmidhuber [63] typically consists of input, forget and output gates as well as a memory cell activation component, as shown in the Figure below 2.7.

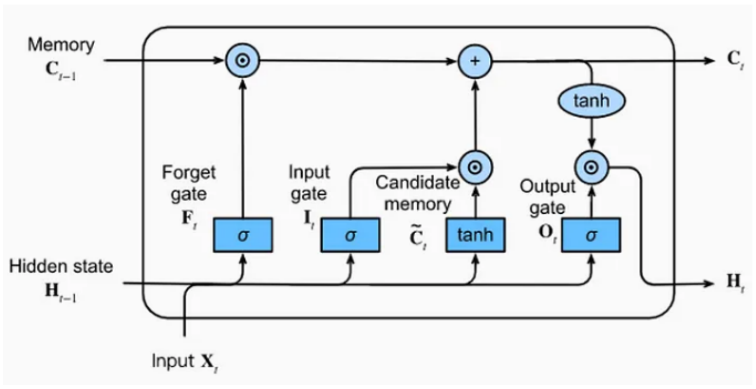

Memory $\mathbf{C}_{t-1}$
$\mathbf{C}_t$
tanh
Forget gate $\mathbf{F}_t$
Input gate $\mathbf{I}_t$
Candidate memory $\tilde{\mathbf{C}}_t$
Output gate $\mathbf{O}_t$
σ
tanh
Hidden state $\mathbf{H}_{t-1}$
$\mathbf{H}_t$
Input $\mathbf{X}_t$

Figure 2.7: LSTM Cell [64]

These gates receive activation signals from various sources and regulate the activation of the cell through certain multipliers. By using these gates, the LSTM can effectively prevent the rest of the network from changing the contents of the memory cells over multiple time steps. Compared to normal RNNs, LSTM networks are able to maintain signal integrity and propagate errors over much longer sequences. This property allows LSTM networks to handle data with complicated and distinct dependencies, making them very effective in various domains where sequence learning is involved [39], [61], [63], [64].

Suppose there are k hidden units, a stack size of n and d inputs. The input $x_t \in \mathbb{R}^{n \times d}$ and the corresponding hidden state of the previous time step $h_{t-1} \in \mathbb{R}^{n \times k}$. The gates in the current time step t are the input gate identified as $g_{\mathrm{t}}^{\mathrm{i}} \in \mathbb{R}^{n \times k}$, the forget gate as $g_{\mathrm{t}}^{\mathrm{f}} \in \mathbb{R}^{n \times k}$,the cell gate as $g_{\text{t}}^{\text{c}} \in \mathbb{R}^{n \times k}$ and the output gate $g_{\text{t}}^{\text{o}} \in \mathbb{R}^{n \times k}$ [62]–[65]. The gates can thus be described as follows:

$$g_{\text{t}}^{\text{i}} = \sigma(W_{Ig^i} x_t + W_{Hg^i} h_{t-1} + W_{g^c g^i} g_{\text{t-1}}^{\text{c}} + b_{g^i}) \tag{2.27}$$

$$g_{\text{t}}^{\text{f}} = \sigma(W_{Ig^f} x_t + W_{Hg^f} h_{t-1} + W_{g^c g^f} g_{\text{t-1}}^{\text{c}} + b_{g^f}) \tag{2.28}$$

As in the previous RNN section, W_{Ig^i} and W_{Ig^f} are the weight matrices for the input layer to the input gate and the input layer to the forget gate respectively, W_{Hg^i} and W_{Hg^f} are the weight matrices from the hidden state to the input gate and from the hidden state to the forget gate. Finally, $W_{g^c g^i}$ and $W_{g^c g^f}$ represent the weight matrices from the cell activation to the input gate and the cell activation to the forget gate. It goes without saying that $b_{g^{o,f}}$ are the biases of both the input gate and the forget gates [39], [62], [64], [65].

$$g_{\text{t}}^{\text{c}} = g_{\text{t}}^{\text{i}} tanh(W_{Ig^c} x_t + W_{Hg^c} h_{t-1} + b_{g^c}) + g_{\text{t}}^{\text{f}} g_{\text{t-1}}^{\text{c}} \tag{2.29}$$

As for the cell activation gate, W_{Ig^c} and W_{Hg^c} are the weight matrices from the input layer and hidden state to the cell gate respectively, b_{g^c} is the cell gate bias [39], [62], [64], [65]. Thusly, the output gate, g_t^o is defined as:

$$g_{\text{t}}^{\text{o}} = \sigma(W_{Ig^o} x_t + W_{Hg^o} h_{t-1} + W_{g^c g^o} g_{\text{t}}^{\text{c}} + b_{g^o}) \tag{2.30}$$

Where, similarly to the input and forget gates, W_{Ig^o} and W_{Hg^o} are the matrices connecting the input layer and hidden state to the output layer, $W_{g^c g^o}$ the weights between cell activation and outputs and b_{g^o} the biases [39], [62], [64], [65]. Finally, the hidden state is defined as:

$$h_t = g_{\text{t}}^{\text{o}} tanh(g_{\text{t}}^{\text{c}}) \tag{2.31}$$

It is necessary to note that the operations between gates are Element wise operations using the Hadamard product operator ($\odot$).

Bidirectional LSTM

In a bidirectional LSTM (BiLSTM), two separate models are used instead of just one. The first model learns the input sequence as it is presented, while the second model learns the inverse of that sequence [66]. This network needs not only the past context but also the future context for the reversed layer. Having two trained models, poses the need to merge or combine them in some way to use their complementary information.

So, in summary, the BiLSTM introduces an additional LSTM layer that works in the opposite direction, effectively reversing the flow of information in the input sequence, as

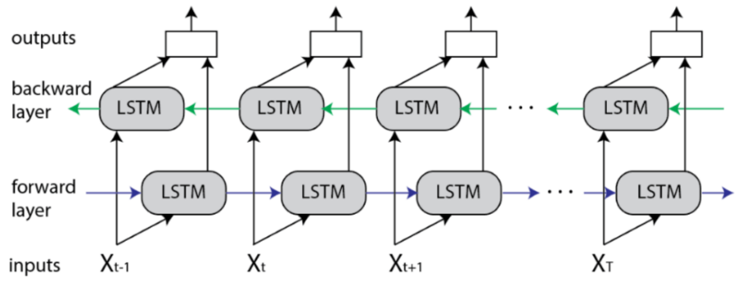


Figure 2.8: Unrolled Bidirectional LSTM [67]

per Figure 2.8. This means that the input sequence is processed in the opposite direction in the additional LSTM layer. Then the outputs of the two LSTM layers are combined using various methods, such as averaging, summation, multiplication or concatenation [66], [67].

2.2.3 Attention Mechanisms

Deep learning uses attention mechanisms to help the model focus on the most important segments of the input data when making predictions. In many scenarios, the input data is very large and complicated, making it difficult for the model to process all the data effectively. Attention mechanisms allow the model to selectively prioritise the crucial aspects of the inputs that contribute most to prediction accuracy, while disregarding the less relevant components. This capability improves the predictive accuracy of the model and the overall performance [68].

In many Deep Learning models, data is processed by passing it through multiple layers of neural networks. These networks consist of many interconnected nodes organised into layers. Each node processes the data and passes it on to the next layer. In this way, the model can extract increasingly complex features from the data as it traverses the network [64], [68], [69]. However, as the data traverses these layers, it can become increasingly difficult for the model to identify the most important information [64], [69], [70]. Attention mechanisms have been introduced to address this limitation. In attention-based models, the model can selectively focus on certain parts of the inputs when making predictions [68]–[70].

Queries, Attention, and Databases

In most conventional networks, the input data must have a well-defined size. Most Convolutional Neural Networks are tuned to a fixed input shape, i.e. ImageNet as an input size of 224× 224 pixels. For most RNNs, the same happens [64], [70]. There are some tricks to mitigate the problems associated with this restriction. For example, in Natural Language RNNs where each variable size is accounted for by processing one token at a time sequentially or by using special convolution kernels [64], [70], [71]. Therefore, with longer sequences, the networks have a hard time remembering everything they have already processed.

A good way to demonstrate attention, as [64] says, is to compare it to a database (D), which, in layman's terms, is a collection of keys (k) and values (v). For a given query (q), only a single value can be represented by a single key. Therefore:

- Queries q, operate in *(k,v)* pairs, are to be valid regardless the size of D.
- The same query q_i can receive different answers, according to the contents of D.
- The "code" being executed in D can be simple.
- D needs not to be simplified in order to make the operations effective.

Thus, the database can be defined as $D = \{(k_1, v_1), ..., (k_m, v_m)\}$ of m tuples of keys and values denoted by a query q. Attention can be defined as:

$$Attention(q, D) = \sum_{i=1}^{m} \alpha(q, k_i)v_i \tag{2.32}$$

Whence $\alpha(q, k_i) \in \mathbb{R}(i = 1, ..., m)$ are scalar attention weights. This operation, as shown in 2.32 is commonly referred to as attention pooling, Figure 2.9. The term "attention" is given to this operation because it focuses specifically on the terms with significant weights (i.e., large weights).

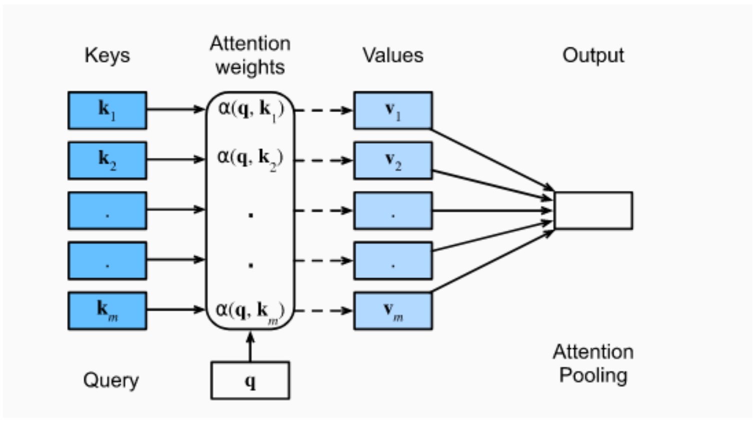


Figure 2.9: The attention mechanism computes a linear combination over values v_i via attention pooling, where weights are derived according to the compatibility between q and k_i [64]

Consequently, the attention over D produces a linear combination of values found in the database. In essence, this encompasses the aforementioned example as a unique scenario where all weights except one are zero. Though there are some special cases:

- $\alpha(q, k_i)$ are non-negative. The output of the attention mechanism is contained in the convex cone spanned by the values v_i.
- $\alpha(q, k_i)$ form a convex combination, $\sum_i \alpha(q, k_i) = 1$ and $\alpha(q, k_i) \geq 0 \ \forall \ i$. The most common setting in deep learning.
- One of the weights $\alpha(q, k_i)$ of the database is 1, while the others are 0. akin to a traditional database query.
- all weights are equal ($\alpha(q, k_i) = \frac{1}{m} \ \forall \ i$). Amounting to averaging across the entire database. Also known as average pooling.

The strategy to ensure that the weights sum to 1 is to normalize them (equation 2.33) and to ensure they are non-negative, exponentiation can be used (equation 2.34).

$$\alpha(q, k_i) = \frac{\alpha(q, k_i)}{\sum_j \ \alpha(q, k_i)} \tag{2.33}$$

$$\alpha(q, k_i) = \frac{exp(\alpha(q, k_i))}{\sum_j \ exp(\alpha(q, k_i))} \tag{2.34}$$

This operation is present in all Deep Learning frameworks and provides desirable properties such as differentiability and a non-vanishing gradient, which are crucial for a model. However, it is important to note that the attention mechanism mentioned above is not the only option available. Some other options are shown below. The really impressive thing is that the code required to perform operations on an extensive set of keys and values, called a query, can be kept remarkably tight. This property makes it desirable for a network layer, as it does not require numerous parameters to learn. Equally convenient is the fact that Attention can effortlessly work with databases of any size without having to change the operations of the Attention pool [64], [70].

2.2.4 Common Attention Mechanisms

The attention mechanism above is not the only one. There are many more mechanisms of attention. A short overview will be given.

Attention Pooling

The attention pooling mechanism is a technique used in Deep Learning models to assign weights or importance to different elements within a given set. It involves calculating a similarity score between a query and each element in the set and then using these scores to determine how much attention or focus to give to each element [72]. Considering Nadaraya and Watson's [73], [74] approach as a kernel density classification and regression problem, its estimators rely on a similarity kernel $\alpha(q, k)$ that relates queries to keys [64], [72]. Some common kernels are:

$$\alpha(q, k) = exp(-\frac{1}{2}||q - k||^2) \tag{2.35}$$

$$\alpha(q, k) = 1 \; if \; ||q - k|| \leq 1 \tag{2.36}$$

$$\alpha(q, k) = max(0, 1) - ||q - k||) \tag{2.37}$$

Regardless of the kernel used, all of them lead to the following equation 2.38, for both regression and classification tasks. If, in case of regression with observations of (x_i, y_i) for features and the respective labels. $v_i = y_i$are scalars and $k_i = x_i$ are vectors. In case of a classification task, one-hot-encoding can be performed on y_i to obtain v_i. The query q denotes the location where f should be evaluated [64], [72].

$$f(q) = \sum_i v_i \frac{\alpha(q, k_i)}{\sum_i \alpha(q, k)} \tag{2.38}$$

Multi-Headed Attention

According to Aston Zhang et al. [64], in order to improve the present model, it may be beneficial to enable the attentional mechanism to integrate insights from different behaviours of the same attentional mechanism, given the same set of queries, keys and values. This includes capturing dependencies with different ranges (i.e. shorter and longer ranges) within a sequence. Therefore, it may be advantageous for our attention mechanism to share different subspaces for representing queries, keys and values.

To achieve this goal, queries, keys and values can be individually transformed using h different linear projections that are learned independently, rather than singular attention pooling. Then, these h projected queries, keys and values are simultaneously input to the attention pooling. Finally, the outputs of h attention pooling processes are concatenated and further transformed by an additional learned linear projection, resulting in the final output (Figure 2.10).

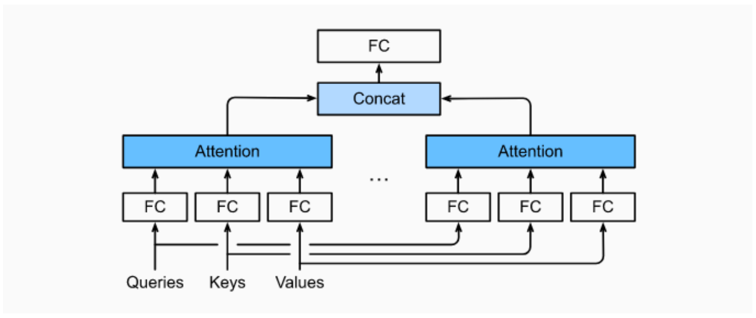


Figure 2.10: Multi-head attention involves concatenating multiple heads and subsequently linearly transforming them [64].

The implementation of multi-head attention can be formally described given a query $q \in \mathbb{R}^{d_q}$, a key and a value $k \in \mathbb{R}^{d_k}$, $v \in \mathbb{R}^{d_v}$ and the corresponding attention head h_i $(i1, ..., h)$. It can be defined as:

$$h_i = f(W_{\mathrm{i}}^{\mathrm{q}} q, W_{\mathrm{i}}^{\mathrm{k}} k, W_{\mathrm{i}}^{\mathrm{v}} v) \tag{2.39}$$

where $W_{\mathrm{i}}^{(\mathrm{q})} \in \mathbb{R}^{p_q \times d_q}$,$W_{\mathrm{i}}^{(\mathrm{k})} \in \mathbb{R}^{p_k \times d_k}$,$W_{\mathrm{i}}^{(\mathrm{v})} \in \mathbb{R}^{p_v \times d_v}$ and f is the attention pooling. The output of the multi-head attention is another transformation via learnable parameters $W_{\mathrm{i}}^{(\mathrm{o})} \in \mathbb{R}^{p_o \times d_o}$ of the concatenation of the heads.

$$W_o \begin{bmatrix} h_1 \\ \vdots \\ h_h \end{bmatrix} \in \mathbb{R}^{p_o} \tag{2.40}$$

Thus,

$$MultiHead(Q, K, V) = Concat(h_1, ..., h_h)W_o \tag{2.41}$$

where each head is given by the equation 2.41. With this design, each head can attend to distinct sections of the input, allowing for the expression of more intricate functions beyond a basic weighted average [64], [70].

Self Attention

Self-attention is used to map a sequence of variable length $x_1, ..., x_n$ to another sequence of the same length, $y_1, ..., y_n$ where $x_i, y_i \in \mathbb{R}^d$ [64], [70]. These are normally the hidden layers within a standard encoder or decoder. Thus, the sequence y_i is described as:

$$y_i = f(x_i, (x_1, x_1), ..., (x_n, x_n)) \in \mathbb{R}^d \tag{2.42}$$

To use self-attention, there are some considerations on the desired data to use. The first factor is the total computational complexity per level. The second factor is the

Layer Type	Complexity	Sequential Operations	Max Path Length
Self Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$

Table 2.2: Comparison of Complexity, Sequential Operations and Maximum Path Length as per Vaswani et al. and Aston Zhang et al. [64], [70]

degree of parallelisation possible, which is determined by the minimum number of sequential operations required. The third factor is the distance between remote dependencies within the network. The ability to learn such dependencies is crucial in sequence implementation tasks. An important aspect that affects this ability is the length of the paths that signals must travel between different positions in the input and output sequences. The shorter these paths are, the easier it is to learn long-range dependencies [70], [75]. Hence, the maximum path length between any two input and output positions in networks (Figure 2.11) comprising various layer types are also compared [64], [70], [75].

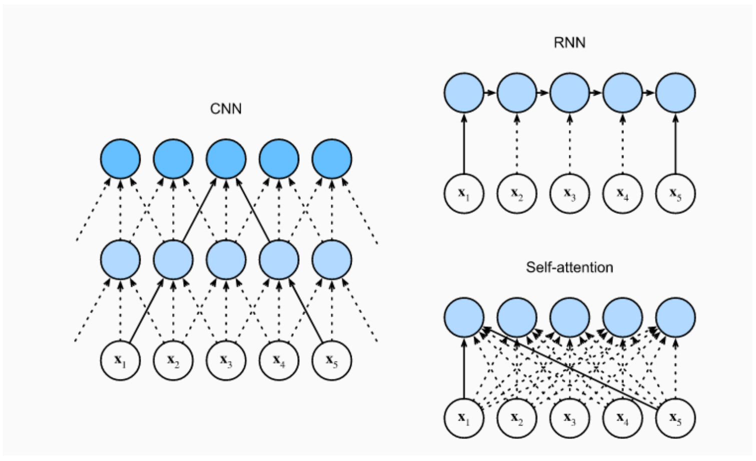


Figure 2.11: Comparison of the CNN, RNN and Self Attention architectures [64]

Comparing the computational complexity, sequential operations and maximum path lengths in the mapping of a sequence of n tokens to another of the same length, where each input/output is represented by a d-dimensional vector, between normal convolutional networks, recurrent networks and self-attention. As indicated in the table 2.2, a self-attention layer makes connections between all positions with a fixed number of sequentially executed operations, while a recursive layer requires $O(n)$ sequential operations. In terms of computational complexity, self-observation layers are faster than recurrent layers when the sequence length n is smaller than the representation dimensionality d [64], [70]. In summary, both

CNNs and Self-Attention benefit from parallel computations and Self-Attention has the shortest maximum path length. Nevertheless, the quadratic computational complexity over sequence length makes Self-Attention excessively slow for extremely long sequences [64], [70].

2.3 Graph Neural Networks

The progress of Deep Learning in various fields is due, in part, to the continued advancement of computational resources such as GPUs, the abundance of large-scale training data, and the ability of Deep Learning to effectively extract underlying representations from Euclidean data [76]. Although Deep Learning is able to capture hidden patterns in Euclidean data, there is a growing demand for applications for data represented in graphs. The complexity of graph data has posed significant difficulties for current machine learning algorithms. Due to the irregular nature of graphs, nodes can vary in size and have no particular order. In addition, nodes within a graph can have a varying number of neighbours, thus, making certain crucial operations challenging to compute [76], [77]. Before talking about Graph Neural Networks (GNNs), it is essential to define a graph. In computer science, a graph is a data structure consisting of vertices (nodes) and edges. The nodes represent the entities and the edges represent the relationships between them [76], [77]. A graph can therefore be defined as $G(V, E)$. Let $v_i \in V$ denote a node and $\{e_{ij}(v_i, v_j) \in E\}$ the edge pointing from v_i to v_j. The adjacency of a node can be defined as $N(v)\{u \in V|(v, u) \in E\}$. And the adjacency matrix A $(n \times n)$ with $A_{ij}1$ if $e_{ij} \in E$ and $A_{ij}0$ if $e_{ij} \notin E$. The graph is attributed, with X input attributes, where $X \in \mathbb{R}^{n \times d}$ is a node feature matrix with $x_v \in \mathbb{R}^d$ representing the feature vector v [76].

Graph theory, tries to define the notion of Node Embedding, which involves mapping nodes to a low-dimensional embedding space (instead of the actual dimension of the graph). This mapping ensures that similar nodes within the graph are positioned close to one another in the embedding space by defining encoders for the nodes which convert the feature vectors (Figure 2.12).

2.3.1 Architecture and Pipeline of the Typical GNN

The general pipeline of a GNN usually starts with the search for a graph structure. There are two different scenarios, a structural and a non-structural one. The structural scenario implies that the structure of the graph is explicit to the application, i.e. molecules,

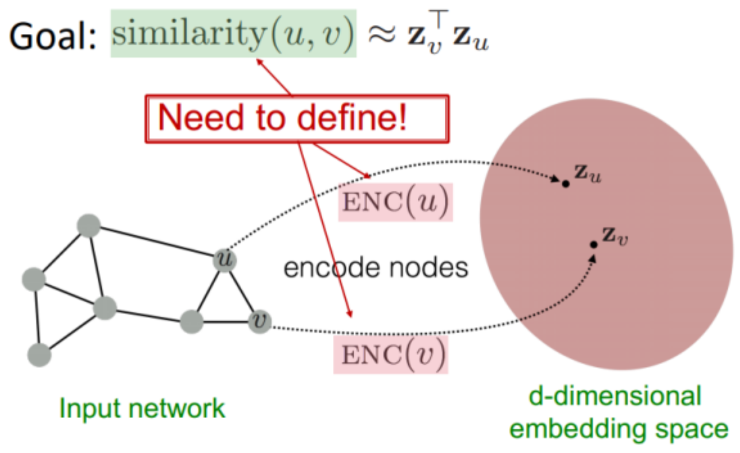


Figure 2.12: Conversion of nodes to feature vectors in a GNN [78]

physical systems, knowledge graphs, etc. In non-structural scenarios, graphs are implicit and must be created from the task at hand [76]. It is then necessary to specify the type and scale of the graph that can be described by these orthogonal categories (can be combined) [77]:

- Directed/Undirected: In directed graphs, edges have a specific direction, indicating a flow of information from one node to another. This directed nature of edges provides additional information compared to undirected graphs. In undirected graphs, each edge can be considered as two directed edges, with information potentially flowing in both directions.

- Homogeneous/Heterogeneous: In homogeneous graphs, nodes and edges have the same type, while in heterogeneous graphs, nodes and edges have different types. The types assigned to nodes and edges are of great importance in heterogeneous graphs and should be carefully considered.

- Static/Dynamic: A graph is said to be dynamic if either the input features or the topology of the graph change over time. It is crucial to handle time information carefully and consider it appropriately in dynamic graphs.

Regarding the scale, there is no concise metric to classify the graphs. The criterion changes according to the speed and memory of the devices they are run in [76], [77], [79]. One should also design the loss function according to the task and training setting.

- Node-level: Node-centric tasks revolve around nodes and include various goals such as node classification, node regression and node clustering. Node classification is about assigning nodes to specific classes, while node regression aims to predict continuous values for each node. On the other hand, node clustering focuses on partitioning nodes into different groups where nodes with similar characteristics are grouped.

- Edge level: Edge-centric tasks consist of edge classification and link prediction. In these tasks, the model must classify different types of edges or predict the presence or absence of an edge between two specific nodes.

- Graph-level: Graph-level tasks include graph classification, graph regression and graph matching. These tasks require the model to learn representations of entire graphs to perform tasks such as categorising graphs, predicting continuous values for graphs, or determining graph similarities and correspondences.

Therefore, with the task type and the training setting (Supervised, Semi-supervised or Unsupervised) a specific loss function for the task can be designed [76], [77], [79], [80].

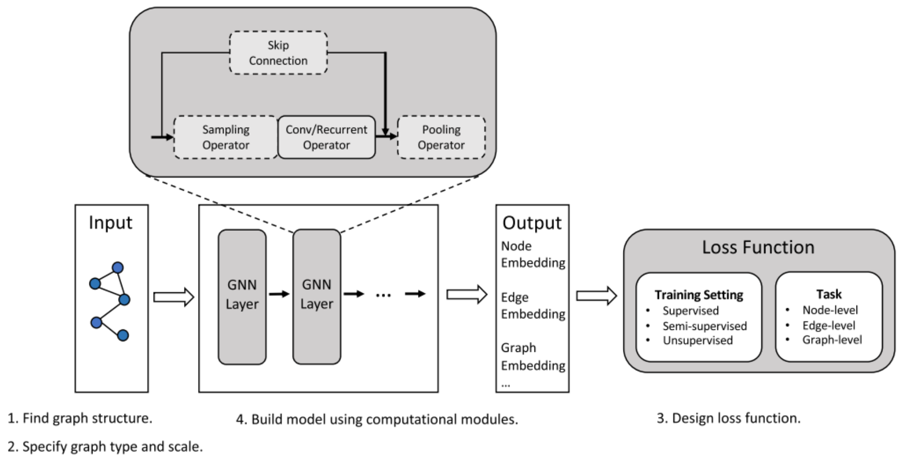


Figure 2.13: Architecture of a GNN [78]

A typical architecture of a GNN model comprises three main modules.

- Propagation Module: The propagation module transfers information between nodes so that aggregated information can include feature-based and topological details. The convolution operators and recurrent operators are commonly used in propagation modules to aggregate information from neighbouring nodes. In addition, the Skip Connection operation is used to gather information from previous node representations and mitigate the problem of over smoothing.

- Sampling module: For large graphs, it is often necessary to use sampling modules to perform propagation efficiently. These sampling modules are usually integrated into the propagation module and enable effective graph propagation while coping with the computational demands of large-scale graphs.

- Pooling Module: When there is a requirement to obtain representations of higher-level sub-graphs or graphs, pooling modules become essential for extracting information from individual nodes. These pooling modules enable the aggregation of node-level information to derive meaningful representations of larger sub-graphs or complete graphs [76], [77], [79].

To this end, by using these modules, a standard Graph Neural Network (GNN) model can be built. Which can be shown by the middle section of the Figure 2.13, depicting a representative architecture of the GNN model.

2.3.2 The Convolutional GNN

There are two main types of GNNs, namely the convolutional GNNs and the recurrent GNNs. In this paper, we will focus on convolutional GNNs because they can generalise the convolution operation from lattice data to graph data [79], which will be of great use in this thesis.

The main goal of Convolutional GNNs (ConvGNNs) is to generate a representation of a node, v, by aggregating its features with those of its neighbours, x_v and x_u, where $u \in \mathbb{N}(v)$. ConvGNNs stack multiple graph convolutional layers to extract a high-level representation of the nodes [76], [77], [79]. These systems solve the issue of cyclic interdependencies by using a fixed number of layers, with each layer having different weights. This architecture allows ConvGNNs to effectively handle cyclic dependencies [76]. ConvGNNs can be divided into two different categories: spectral-based and spatial-based [76], [77], [79], [80].

Spectral-Based ConvGNNs

Spectral-based approaches define graph convolution by adopting filters based on graph signal processing principles. In this context, the graph convolution operation is understood as a means of removing noise from graph signals [76], [77], [79]–[81]. These are based on graph signal processing, assuming that graphs are undirected [76], [81]. The graph Laplacian matrix of an undirected graph is described as $LI_n - D^{\frac{1}{2}}AD^{\frac{1}{2}}$, where D is the diagonal matrix of node degrees, $D_{ij} \sum_j (a_{i,j})$. Since the Laplacian is symmetric, positive and semi-defined [81], it can be factorised as $LU\Lambda u^T$, where $U(u_0, u_1, ..., u_n - 1) \in \mathbb{R}$ is the ordered eigenvalue matrix and Λ is the diagonal matrix of eigenvalues $\Lambda_{ij}\lambda_i$. The eigenvalues of the normalised Laplacian form an orthonormal space U^tUI [76], [77], [80], [81]. In graph signal processing, the signal x is transformed using the graph Fourier transform $\mathcal{F}$, the convolution operation is performed and the signal is transformed back using the Fourier inverses.

$$\begin{aligned} \mathcal{F}(x) &= U^T x \\ \mathcal{F}^{-1}(x) &= Ux \end{aligned} \tag{2.43}$$

and the graph convolution of the signal x with a filter $g \in \mathbb{R}^n$ can be described as:

$$\begin{aligned} x * g &= \mathcal{F}^-1(\mathcal{F}(x) \odot \mathcal{F}(g)) \\ &= U(U^T x \odot U^T g) \end{aligned} \tag{2.44}$$

where $\odot$ denotes the elementwise product and a filter g as $g_\theta = \text{diag}(u^t g)$. The equation 2.44 can be further simplified as:

$$x * g_\theta = Y g_\theta U^T x \tag{2.45}$$

as *Bruna et al.* [82] states, one must assume the filter is a set of learnable parameters and considers signals with multiple channels. Thus, the graph layer of a spectral CNN can be defined as:

$$H^{\text{k}}_{\text{i,j}} = \sigma(\sum_{i=1}^{f_{k-1}} U\Theta^{\text{k}}_{\text{i,j}} U^T H^{\text{k-1}}_{\text{i,j}})\ (j = 1, 2, ..., f_k) \tag{2.46}$$

where the diagonal filter matrix of learnable parameters is assumed as $g_\theta = \Theta^{\text{k}}_{\text{i,j}}$, k the layer index, $H^{k-1} \in \mathbb{R}^{n \times f_{k-1}}$ the input graph signal, $H^0 = X$, f_{k-1} the number of input channels.

There are many improvements of this type of network like the Chebyshev spectral CNN (ChebNet), which approximates the filter by Chebyshev polynomials of the matrix of eigenvalues, and the CayleyNet which further applies the Cayley polynomials to capture narrow frequency bands [76], [77], [81], [82].

Spatial-Based ConvGNNs

Spatial-based define graph convolutions by information propagation, analogous to the convolutional operation of a conventional CNN on an image, this method defines graph convolutions based on the node's spatial relations [76], [77]. The basic spatial approach defines convolutions directly on the graph based on its topology. The difficulty arises when defining convolution operations with differently sized neighbourhoods while maintaining the local invariance of CNNs [76], [77], [81]. Hence, there are various methods to describe the convolutions. One of the simpler ways to do this is by using Duvenaud et al. [83] Neural FPs, which use different weight matrices for nodes with different degrees [76], [83]:

$$\begin{gathered} t = h_{\mathrm{v}}^{\mathrm{t}} + \sum_{u\in N(v)} h_{\mathrm{u}}^{\mathrm{t}} \\ h_{\mathrm{v}}^{\mathrm{t+1}} = \sigma(tW_{|\mathrm{N(v)}|}^{\mathrm{t+1}}) \end{gathered} \tag{2.47}$$

where $W_{|\mathrm{N(v)}|}^{\mathrm{t+1}}$ is the weight matrix for nodes of $|N(v)|$ at layer $t+1$. Unfortunately, this method cannot be applied to large-scale graphs.

The Neural Network for Graphs (NN4G) [84], is probably the first work towards Spatial-based ConvGNNs. NN4G employs a compositional neural architecture with independent parameters at each layer to learn the mutual dependency among graph elements. The architecture is incrementally constructed to extend the neighbourhood of a node. Graph convolutions in NN4G are accomplished by directly summing up the neighbourhood information of a node. Additionally, NN4G incorporates residual connections and skip connections to retain and memorize information throughout each layer. Accordingly, the next layer node states can be described as [76], [84]:

$$h_{\mathrm{v}}^{\mathrm{k}} = f(W^{k^T} x_v + \sum_{i=1}^{k-1} \sum_{u\in N(v)} \Theta^{k^T} h_{\mathrm{u}}^{\mathrm{k\text{-}1}}) \tag{2.48}$$

where $f(\cdot)$ is the activation function and $h_{\mathrm{v}}^{0} = 0$. Written in matrix form, it is defined as:

$$H^k = f(XW^k + \sum_{i=1}^{k-1} AH^{k-1}\Theta^k) \tag{2.49}$$

Another improvement is the Attention-based spatial approaches, which have been successfully used in many sequence-based tasks [70], [85], [86].

The graph attention network (GAT) [87], incorporates the attention mechanism into the propagation step, which is a characteristic of this approach. It calculates the hidden states of each node by attending to its neighbours, utilizing a self-attention strategy. The hidden state of a node can be defined as:

$$h_{\mathrm{v}}^{\mathrm{k}} = \sigma(\sum_{u\in N_v \cup v} \alpha_{\mathrm{v\,u}}^{\mathrm{k}} W^k h_{\mathrm{u}}^{\mathrm{k\text{-}1}}) \tag{2.50}$$

$$\alpha_{\mathrm{v\,u}}^{\mathrm{k}} = softmax(LeakyReLU(a^T[W^k h_{\mathrm{v}}^{\mathrm{k\text{-}1}} || W^k h_{\mathrm{u}}^{\mathrm{k\text{-}1}}])) \tag{2.51}$$

$$softmax = \sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K}} \; for i = 1, ..., K \, and \, z = (z_1, ..., z_k) \in \mathbb{R}^K \tag{2.52}$$

Summarily, the weight matrix, denoted as W, represents the linear transformation applied to each node. Additionally, a refers to the weight vector of a single-layer MLP (Multilayer Perceptron). The softmax function guarantees that the attention weights, when summed, equal one across all neighbours of the node v. Additionally, GAT (Graph Attention Network) incorporates multi-head attention to enhance the model's ability to capture complex patterns and relationships.

3 Related Work

The main focus of this thesis will be on architectures capable of learning and processing time sequences without the need for a time warping methods like previously mentioned. Therefore, this thesis will focus mainly on Two distinct types of architectures (RNNs and GNNs) in two different tasks. Action recognition (a classificatory task) and motion prediction (estimation task).

3.1 Human Motion Models

Human motion models are mathematical or computational representations that aim to capture and simulate the movement patterns and behaviours of human beings. These models can describe various aspects of human motion, including locomotion, gestures, postures, and interactions with the environment. Human motion models are commonly used in fields such as computer animation, robotics, biomechanics, virtual reality, and motion analysis to simulate, analyse, or predict human movements.

3.1.1 RNN Based Classification

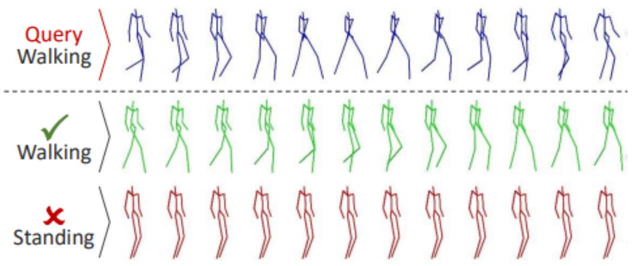


Figure 3.1: Measurement of the similarity to a query sequence (Walking) as per [88]

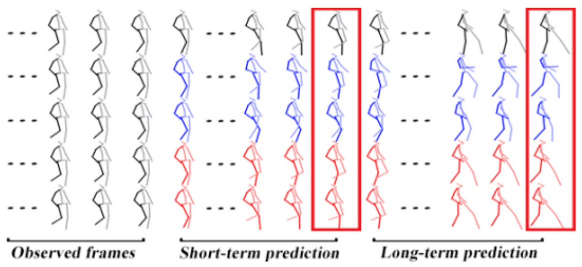


Figure 3.2: Motion prediction [24].

When comparing human poses over a series of frames, determining the similarity between two sequences of poses or motion is a challenging problem. This is because human motion tends to exhibit varying patterns across different sequences, resulting in specific pose configurations appearing at different time points, even for the same type of human motion [88]–[90]. In the Figure 3.1, the provided sequences represent two actions that fall within the same class. Additionally, these sequences exhibit variations in length, meaning they consist of a different number of frames.

In addition, these sequences also vary in length, meaning they can have a different number of frames. This aspect further complicates the establishment of a universal similarity measure. However, despite the challenges, accurately estimating the similarity between human poses across a sequence is an essential step in various human motion analysis tasks, including action retrieval and recognition [88]–[91].

Metric learning and human motion

Traditional methods utilized for comparing human motion sequences typically rely on estimating the L2 displacement error or using Dynamic Time Warping (DTW). The L2 displacement error calculates the squared distance between corresponding joints in two sequences at a specific time instant. However, as Martinez et al. [23] demonstrated, this measure often overlooks the unique motion characteristics, as a repeated constant pose in a sequence may erroneously be considered a better match to a reference sequence than a visually similar motion with distinct temporal evolution [23], [88]. In contrast, as shown by Keogh et al. and Vintsyuk [92], [93], Dynamic Time Warping (DTW) attempts to address this issue by warping the two sequences through compressions or expansions, aiming to optimize the alignment between local poses. However, DTW can encounter challenges in accurately estimating similarity when there are minor temporal variations in the motion dynamics, such as small variations in peaks and plateaus demonstrated by Keogh et al.'s work [88], [93].

According to Coskun et al. [88], early works on time series metric learning measure the similarity as a two-step method [94], [95]. Initially, the model identifies the optimal alignment between two time series and subsequently calculates the distance using the aligned series. Typically, this alignment is determined using the DTW transform, where all possible alignments are evaluated and ranked based on a manually designed local metric [88], [94], [95]. These approaches suffer from two primary limitations. On the one hand, the models are very complex, with a big O notation of $O(n^2)$. On the other hand, and probably the most important, the local metric struggles to capture relationships within high-dimensional data [88]. In order to tackle these issues, recent studies have concentrated on finding a low-dimensional embedding that can effectively measure the distance between time series [20], [96]–[98]. However, the objective of metric learning is still the same. In essence, the objective is to acquire an embedding for human motion sequences that enables a similarity metric to be established between two sequences of human motion, X and Y where:

$$\begin{aligned} X &:= \{x_1, ..., x_n\} \\ Y &:= \{y_1, ..., y_m\} \end{aligned} \tag{3.1}$$

In which, x_t and y_t represent the poses at time-step t. These can be expressed as the Euclidean distance squared in the embedding space:

$$d(f(X), f(Y)) = ||f(X) - f(Y)||^2 \tag{3.2}$$

where $f(\cdot)$ is the learned embedding function that maps the variable length sequence, and $d(\cdot, \cdot)$ the squared Euclidean distance [88].

3.1.2 Recent improvements

Deep metric learning methods began with Siamese architectures [88], [96], [97] that minimize a contrastive loss[88], [99]. This loss function, *f*, works by minimizing the squared Euclidean distance $d(f(X), f(Y))$. The aim is to learn an embedding for human motion sequences that minimizes the similarity metric when X and Y belong to the same category, and otherwise, maximizing it:

$$L_{contrastive} = (r)\frac{1}{2}d + (1-r)\frac{1}{2}[\max(0, \alpha_{margin} - d)] \tag{3.3}$$

where $r \in \{1, 0\}$ indicates whether X and Y belong to the same category or not, and α_{margin} determines the margin between samples from different categories. During the training process, the contrastive loss function penalizes cases where samples from different categories are closer than the threshold α and when samples from the same category have a distance greater than zero. However, this equation indicates that the contrastive loss only considers pairwise relationships between samples, thereby partially leveraging the relative relationships among categories [88].

The FaceNet architecture [100], proposes the utilization of triplet loss for learning facial embeddings in tasks such as facial recognition and verification. It demonstrates superior performance compared to contrastive loss in feature learning. However, the process of searching for hard-negative samples becomes computationally inefficient as the training set and the number of different categories increase. Subsequently, recent research has mainly concentrated on meticulously constructing batches and utilizing all samples within the batch. Triplet learning is more effective in utilizing these relationships by considering three samples simultaneously, where the first two belong to the same category and the third belongs to a different category. Recent research has demonstrated that leveraging the relative relationships among categories is crucial for achieving a high-quality learned embedding [88], [99]. The triplet loss imposes an *alpha* distance on embedding samples from the same category compared to samples from a different category. Let X, X^+ and X^- be three human motion samples [88], [101]. Then:

$$L_{triplet} = \max(0,\ ||f(X) - f(X^+)||^2 - ||f(X) - f(X^-)||^2 + \alpha_{margin}) \tag{3.4}$$

where X, X^+ are the same category samples and X^- the sample from a different category. Normally in the literature, X is commonly known as the anchor and X^+, x^- the positive and negative samples respectively [88], [101]–[104].

One issue with triplet loss is the parameterization of α_{margin}. This can be overcome using Neighbourhood Components Analysis (NCA) as per Roweis et al. [104] can be described as:

$$L_{NCA} = \frac{exp(-||f(X) - f(X^+)||^2)}{\sum_{x^- \in C} exp(-||f(X) - f(X^-)||^2)} \tag{3.5}$$

where C represents all the categories except the one of the positive sample [88], [104]. In an ideal scenario, when considering triplets of samples, it is expected that samples from the same category will be clustered together in the embedding space. Nevertheless, studies have revealed that the majority of formed triplets do not provide informative training signals, and exploring all possible triplet combinations is not feasible. Consequently, the model will be trained using only a limited number of informative triplets [88], [100]. Thus, most recent networks for action recognition tend to work on these principles of loss and embedding.

3.1.3 RNN Based Prediction

The task of learning statistical models of human motion is challenging due to factors such as high dimensionality, non-linear dynamics, and the stochastic nature of human movement [23]. RNNs are neural network models designed to handle sequential data by utilizing recurrent connections that connect the neural activations at consecutive time steps [105]. Consequently, these networks seem to be good candidates for motion estimation (Figure 3.2).

Motion Generation and Modelling

The generation of realistic human motion using probabilistic models trained on motion capture data has previously been explored within the domains of computer graphics and machine learning [105]. The majority of research has concentrated on expanding latent-variable models that adhere to state-space equations, such as hidden Markov models. These approaches explore the balance between model capacity and inference complexity. Non-linear motion prediction can also be achieved using Gaussian Processes Dynamical Models, which also facilitate the learning of temporal dynamics through expectation maximization [20], [23], [24], [105]–[107].

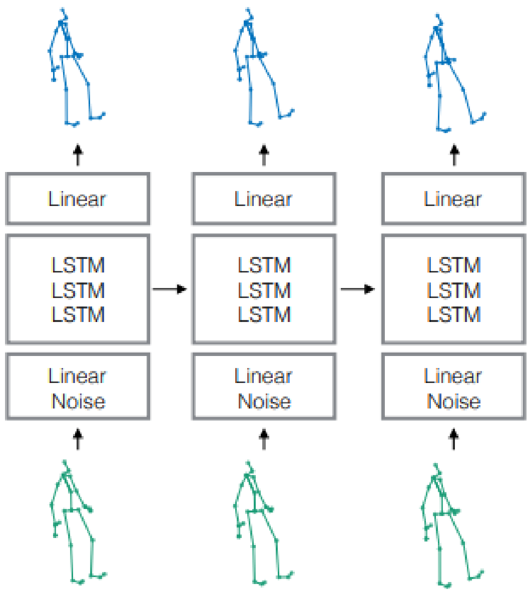


Figure 3.3: LSTM3LR architecture

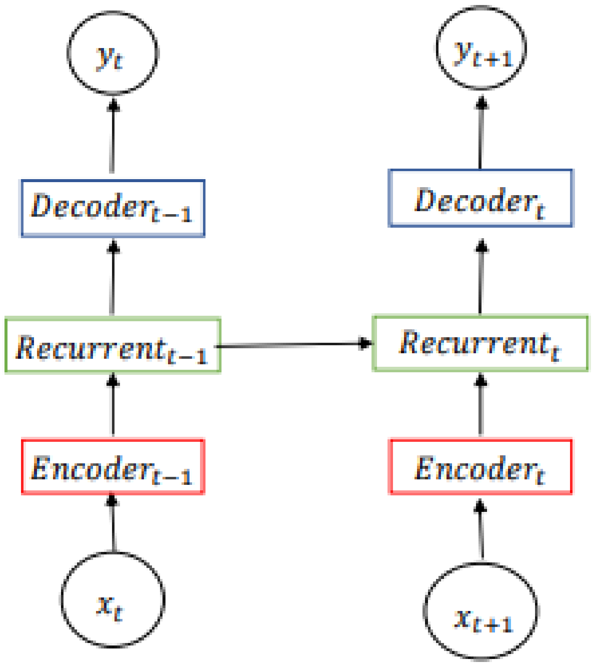


Figure 3.4: ERD based on the LSTM3LR architecture.

Figure 3.5: Fragkiadaki et al.'s work in RNNs for motion recognition [23], [105]

While these models have demonstrated their effectiveness in capturing simple motions, they often fall short compared to deep learning models when dealing with more complex motions and longer prediction tasks. RNNs have, for motion prediction, proven to be very successful in sequence-to-sequence (Seq2Seq) tasks. Fragkiadaki et al. [105] proposed two architectures (Figure 3.5): LSTM-3LR and Encoder-Recurrent-Decoder (ERD). Both models utilize concatenated LSTM units, but the latter incorporates non-linear space encoders for data preprocessing. Fragkiadaki et al. [105] also acknowledges that, during the inference process, the network tends to accumulate errors and generate unrealistic human motion rapidly. As a solution, they suggest gradually introducing noise to the input during the training phase [23], [105], [107].

In a more recent architecture, Jain et al. [26] presented a new method called structural RNNs (SRNNs). This approach involves utilizing a manually constructed graph that represents the semantic knowledge of the RNN. By doing so, a bi-layer architecture is created, where each individual RNN unit is assigned to parts of the data that are semantically similar. Additionally, the authors incorporated a noise scheduling technique introduced by Fragkiadaki et al. [23], [24], [105].

Also, a noteworthy mention is the introduction of the exponential map parameterization of each joint in a kinematic tree. This was achieved using code provided in the Human 3.6M dataset, by Ionescu et al. and Jain's preprocessing [1], [26]. This parameterization of the lie group theory managed to yield better results than the normal Cartesian joint representation.

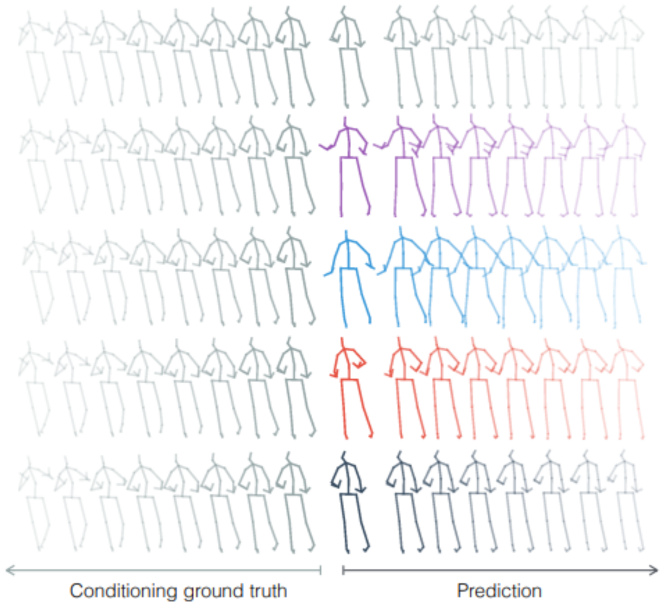


Figure 3.6: Martinez et al. implementation of short time prediction [23]

Martinez et al. [23], improved this concept by using a (Seq2Seq) architecture which addresses the short-term prediction as the search for a function that maps an input sequence to an output sequence [23]. Seq2seq employs a dual network setup: an encoder network which accepts the inputs and generates an internal representation (Figure 3.6), and a decoder network that utilizes the internal state to generate a prediction based on the maximum likelihood estimate [23], [108]. One advantage of this architecture is that the training process closely resembles the protocol used during testing, particularly in terms of the encoding-decoding procedure [23].

3.1.4 Graph Based Motion Prediction

As previously stated in the background section, GCNs extend the convolution operation to data that is structured according to a graph [106].

Graphs have emerged as a suitable option for depicting the human body, predominantly created manually by capitalizing on the inherent arrangement of the kinematic tree and encoded through Graph Convolutional Networks (GCN) [25], [26], [106], [107], [109]. Yan et al.'s [109] approach involves learning the adjacency matrix of the graph, but it remains confined to the connectivity of the kinematic tree [25]. In more recent studies, researchers have investigated the interconnection of all joints, enabling the learning of graph edges [25], [106]. Similarly, this approach allows the training process to determine the graph's connectivity and edge weights in a data-centric manner.

Sofianos et al.[25], complemented by the work of Szegedy et al. and Bütepage et al. [110], [111] proposes the usage of separable convolutions. The aim is to separate the processing of cross-channel correlations and spatial correlations by utilizing 1x1 convolutional filters for the former and channel-wise spatial convolutions for the latter. This approach involves using depthwise separable convolutions, which assume that the cross-channel and spatial correlations can be sufficiently decoupled. Therefore, it is considered more favourable not to combine them in the mapping process [25], [110], [111].

Consequently, as shown in Figure 3.7, a sample of the spatial and temporal adjacency matrices, A^s (left) and A^t (right). The red dots represent the human body joints of the Human3.6M dataset [1] where the learnt joint-joint relations mainly follow the kinematic tree, but also manage to denote some long-term connections (foot-foot or foot-hand). The matrix A^t demonstrates the transfer of information from preceding to subsequent frames, indicated by higher absolute values located in the bottom-left region.

With GNNs still in mind, Sirui et al. [102], developed an anchor-based spatial-temporal network. This network incorporates the crucial understanding that forthcoming movements are not entirely arbitrary or unrelated to one another. Instead, they possess deterministic characteristics aligned with physical laws and human body limitations while continuing patterns from past movements (Figure 3.8). To illustrate, the network anticipates that certain future motions will share deterministic changes in velocity or direction, although

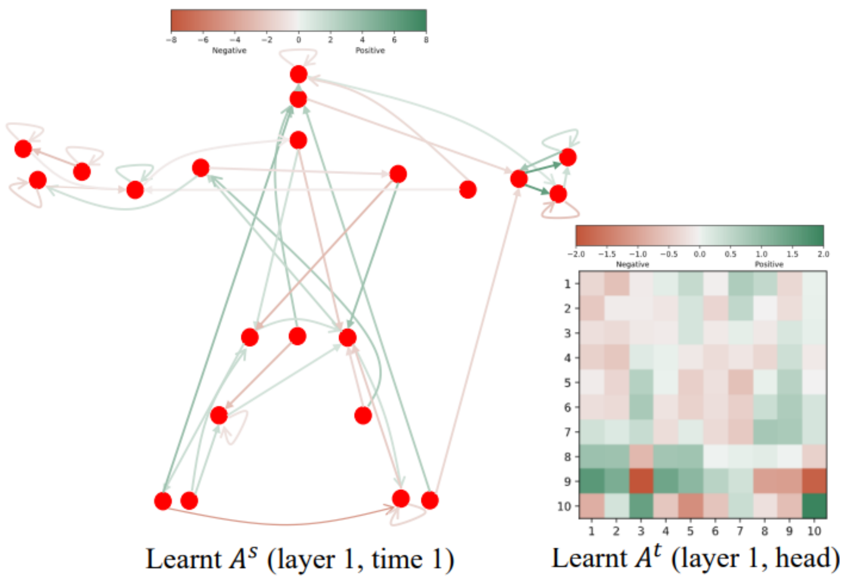


Figure 3.7: Learnt joint relations following a kinematic tree and other connections between joints [25]

their magnitudes may vary stochastically [102].

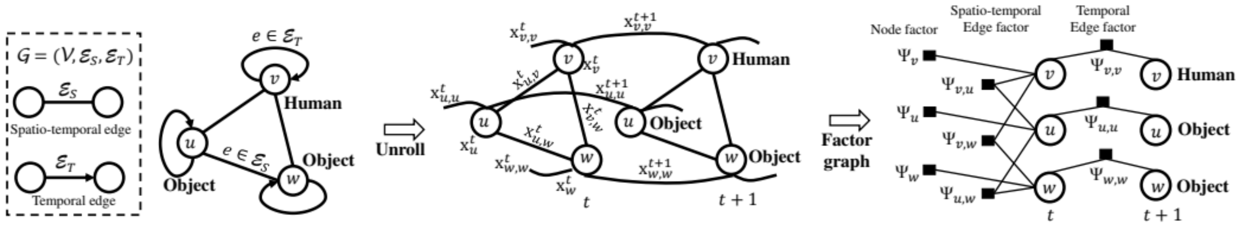


Figure 3.8: An example spatio-temporal graph in the human activity context [26]

3.1.5 Considerations on the Transformer Architecture

It is noteworthy to mention why the ever-so-popular transformer architecture wasn't chosen. Transformers have been very popular among natural language applications and some computer vision tasks. Due to the problem at hand, the usage of a joint-based skeleton in a kinetic tree context, graph networks are a natural choice for this. These encode the tree via a learnable adjacency matrix of the graph, exploring all the interconnected joints, allowing for the network to learn a data-driven graph connectivity and edge weights [25]. If the movement data can be naturally represented as a graph, GNNs may be a better choice due to their ability to model the relationships and dependencies between entities. On the other hand, if the movement data can be effectively framed as a sequential sequence, Transformers can be suitable, leveraging their ability to capture long-range dependencies and global contextual information. It is important to consider the specific characteristics of the data and experiment with both architectures to determine which one performs better for the given movement prediction task [112]–[115]. Thus, some characteristics of graph neural networks for motion prediction are:

- GNNs are well-suited for movement prediction when the data can be represented as a graph, such as tracking objects in a scene or predicting the movement of nodes in a network.
- GNNs can capture the dependencies and relationships between entities in the graph, allowing them to model the dynamics and interactions that influence movement patterns.
- GNNs can incorporate information from the local neighbourhood of each node, taking into account the spatial context and previous movements of neighbouring entities.
- GNNs have been successful in various movement prediction tasks, including pedestrian trajectory prediction, vehicle trajectory prediction, and animal movement analysis [25], [87], [112], [113], [116].

The transformer architecture for movement prediction:

- Transformers are commonly used for sequential data, such as natural language text or time series data. If the movement data can be framed as a sequential sequence, Transformers can be applied.
- Transformers effectively capture long-range dependencies and model complex patterns in sequential data, which can be relevant for movement prediction. They also excel in tasks where contextual information and global dependencies across the sequence are crucial, such as language modelling and machine translation.
- Transformers have been adapted for trajectory prediction tasks by converting the movement data into a sequential format and using self-attention mechanisms to capture temporal dependencies [117]–[119].

3.2 Skeleton Parameterization

Before going into a detailed analysis of the proposed method's architecture and pipeline, a brief summary of the data space needs to be given, for there are two main approaches with their pros and cons. Basic 3D XYZ Cartesian and 3D ZXY Exponential map format.

A more detailed overview of the Lie Theory can be found in the Appendix A.

3.2.1 Exponential Mapping and Lie Algebra

Exponential mapping is a mathematical technique used to represent rotations or transformations in a compact and efficient manner. It is particularly useful for representing human motion in robotics and computer graphics [120]. In exponential mapping, a rotation or transformation is represented as an exponential function of a special kind of matrix called "Lie group". Lie algebra captures the infinitesimal rotational displacements or velocities associated with the rotation. The exponential function "maps" the Lie algebra to the corresponding rotation or transformation in the Lie group. By using it, rotations can be interpolated linearly between different poses or motion states. This allows for smooth and natural-looking transitions between poses, making it suitable for motion prediction and animation [20], [120], [121]. Hence, for human motion prediction, lie algebra and exponential mapping have some benefits:

- Provides a compact and efficient way to represent the motion of a human body. It uses a minimal number of parameters to describe the pose or movement, which makes it suitable for storing and sharing motion data.

- Allows for linear interpolation between different poses or motion states. This linearity property makes it easier to generate smooth and natural-looking transitions between poses, which is desirable in motion prediction.

- Describes smooth continuous transformations. Human body movements can be represented as transformations on a manifold, and the use of exponential mapping allows for a consistent and accurate representation of these transformations.

- compatible with the underlying kinematics of human motion. It accurately represents the rotational movements of joints and the constraints imposed by the skeletal structure. This makes it well-suited for modelling and predicting human motion.

- Provides numerical stability when performing computations involving rotations or transformations. It avoids singularities and numerical instabilities that can arise with other representations, such as Euler angles, quaternions, or rotation matrices [20], [120]–[124].

Even though there are a lot of advantages to the usage of exponential mapping, there are some valid points against its use. Two of the more valid arguments are the associated complexity in data transformation and the non-intuitivity of the parameters:

- According to the literature [121], [124], exponential maps require complex mathematical operations and transformations which are difficult to implement and understand. Bound to this, the implementation of forward and inverse kinematics are not easy tasks to implement, requiring a lot of previous knowledge and assumptions like the distance between points.

- The parameters themselves used in exponential maps do not have a direct physical correspondence. Due to the lack of intuitive mapping, the motions become hard to interpret and even harder to manipulate in the representation without the help of forward and inverse kinematic functions.

Another downside of the exponential map is the increased need for computational power and complexity. As aforementioned, these transformations require some level of computational power. Thus, the computation of the transformations and interpolations can be expensive, especially when dealing with medium to large amounts of data.

4 Developed Work

In the previous two chapters, an overview of the literature pertaining to human motion detection and human action recognition was given, as well as the reasoning and the basics of how the models of these architectures work. The highlighted studies in the Related Work chapter show that there have been some attempts to predict and classify movement in the machine learning and computer vision fields in the span of a decade, but there are some inherent difficulties associated with it.

From the aforementioned architectures, the most successful in the short to medium term prediction have been the Graph Neural Networks, these have an easier time converting a kinematic tree to the graph notation for motion prediction. Jointly, the proposed method also incorporates an LSTM-based action recognition model capable of high-accuracy prediction of a given sequence.

4.1 Pipeline of the Proposed Approach

The whole of the approach narrows down to three main actions to take, as seen in Figure 4.1. First, there is the need to convert a sequence of images into a sequence of parameterized virtual, joint-based skeletons. This can be achieved by many off-the-shelf estimators like *MediaPipe* or *OpenPose*.

After obtaining the skeleton representation in a 3D Cartesian point cloud or vectors of 3D Euler angles, the goal is to estimate the short-term prediction of the next T frames (5 to 20 frames) with the help of an average model that is capable of consistently estimate the correct next few poses with no temporal interference, i.e. the speed of the exercise should not matter to the next T frames. The network should be able to compensate the differences in speed. It is expected that the model can not be inferred in real time due to the lack of computational power in most common machines. The predicted poses will then be compared with the ground-truth (the observed poses of the patient in the next T frames)

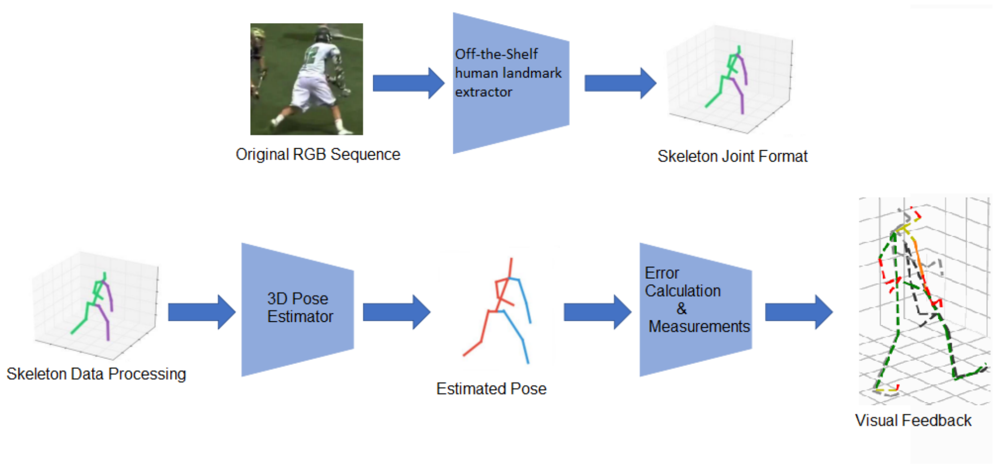


Figure 4.1: High level pipeline of the proposed method.

and the Euclidean distance between the real and predicted joints calculated to give feedback to the user. Finally, a action recognition classification model will be inferred to give the feedback of the overall exercise (*Well done, Not well done*).

4.1.1 Experimental Datasets

For training and testing, a few datasets were used. The Human3.6M [1] which has a great baseline of poses for action recognition with 32 joints denoting 17 scenarios already parameterized in 3D Cartesian poses, 3D Euler and raw angles provided by a Vicon Nexus mocap system. Another great dataset also used was the AMASS dataset [2].

Finally, we used a dataset from previous work done by the ISR, back in 2019, called "The PROZIS challenge" project in partnership with the company PROZIS [125], [126]. The dataset has some fitness workout exercises which are annotated with several types of possible errors. Unfortunately, this dataset was parameterized to skeleton joints but with no additional information on joints indexes or the corresponding kinematic tree, so it was just used for action recognition tasks where the visualization was not needed.

4.2 Action Recognition

For the action recognition aspect of the developed work, Coskun et al.'s [88] approach of a BiLSTM with a self attention mechanism was chosen. This method already had some internal research made on it by previous INPACT work.

4.2.1 Coskun's Self Attentive LSTM

Some of the basics of the inner workings of the most common action recognition networks were already given in the related work and background sections. Coskun's approach takes some inspiration on some already discussed topics like Neighbourhood Components Analysis (NCA), BiLSTMs and attention mechanisms and improves on them. One of the first improvements was on the loss function used. As previously mentioned, this architecture uses a variation of NCA, based on the Maximum Mean Discrepancy (MMD).

MMD-NCA

As Coskun et al. [88] states, MMD, calculates the divergence between two distinct distributions, p and q. It does so by considering the dissimilarities of the average embeddings in Hilbert spaces. This can be expressed as follows:

$$\text{MMD}[k,p,q]^2 = ||\mu_q - \mu_p||^2 = E_{x,x'}[k(x,x')] - 2E_{x,y}[k(x,y)] + E_{y,y'}[k(y,y')] \tag{4.1}$$

where x and x' are drawn independent and identically distributed random variables (IID) from p while y and y' are IID from q while k represents the kernel function and E the expectation operator.

$$k(x,x') = \sum_{q=1}^{K} k_{\sigma_q}(x,x') \tag{4.2}$$

In the equation 4.2 k_{σ_q} denotes a Gaussian kernel with a bandwidth parameter σ_q, while K denotes the hyperparameter, number of kernels. If the expected values are replaced by the given samples, then:

$$\text{MMD}[k,X,Y]^2 = \frac{1}{m^2}\sum_{i=1}^{m}\sum_{j=1}^{m} k(x_i,x'_j) - \frac{2}{mn}\sum_{i=1}^{m}\sum_{j=1}^{m} k(x_i,y_j) + \frac{1}{n^2}\sum_{i=1}^{m}\sum_{j=1}^{m} k(y_i,y'_j) \tag{4.3}$$

Where $X := \{x_1, ..., x_m\}$ is the sample set from p and $Y := \{y_1, ..., y_n\}$ the sample set from q, thus, the equation 4.3 allows the measurement of the distance between the distribution of the two sets. The loss function is designed then to encourage the network to minimize the gap between the distribution of anchor samples and positive samples, while simultaneously maximizing the gap with the distribution of negative samples. Rewriting the NCA equation (3.5) in a different form when considering a specified quantity of N anchor-positive sample pairs, as $\{(X_1, {}^{+}_{1}), ..., (X_N, {}^{+}_{\text{N}})\}$ and $N \times M$ negative samples from M different categories $C = \{c_1, ...c_M\}$ as $\{{}^{-}_{\text{c1,1}}, ..., X^{-}_{\text{c1,N}}, ..., X^{-}_{\text{cM,N}}\}$ then:

$$L_{MMD-NCA} = \frac{exp(-MMD[k, f(X), f(X^{+})])}{\sum_{j=1}^{M} exp(-MMD[k, f(X), f(X^{-}_{\text{cj}})])} \tag{4.4}$$

whence X and X^+ denote motion samples belonging to the same category, whereas X_{cj} represents samples from the category $c_j \in C$. The single update includes M distinct negative classes that are randomly selected from the training data.

The MMD-NCA loss, proposed by Coskun et al. [88], effectively reduces the overlap between category distributions in the embedding. Simultaneously, it ensures that samples from the same distribution are kept as close as possible.

Architecture

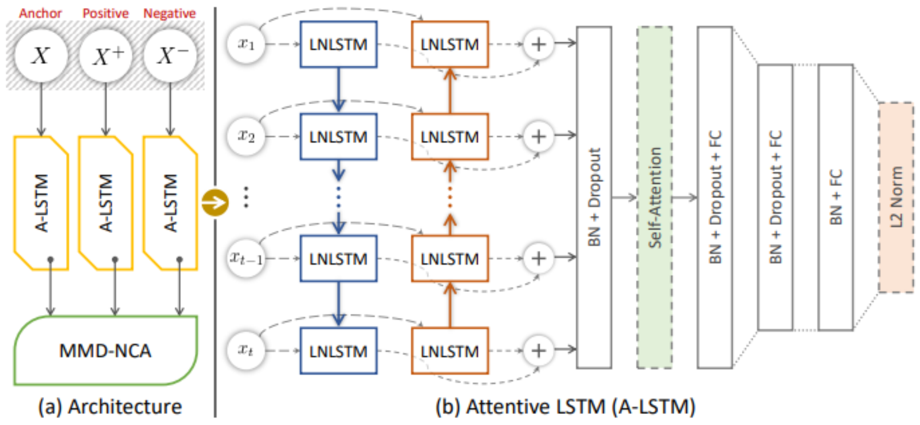


Figure 4.2: Coskun et al.'s Network. [88] a) the architecture itself. and b) the attention based model that uses layered normalization

As presented in Figure 4.2, the model consists of two distinct parts, the BiLSTM and the attention mechanism the author's reasoning to use LSTMs is to overcome the vanishing gradient problem of RNNs, showing that LSTMs can, indeed, capture long term dependencies better.

Layer normalization

According to Coskun et al. and other authors [127]–[129], batch normalization is a key player in the triplet model's accuracy. Nevertheless, the direct implementation of LSTM architectures may lead to a reduction in the accuracy of the model [88], [130]. Consequently, Coskun et al. [88] proposed the usage of layer normalized LSTMs. Looking back at the equations of the background's LSTM section (2.2.2) and assuming that n steps of motion $X = (x_1, ..., x_n)$ are given, a simpler version of the layer normalized LSTM can be described by the following equations:

$$g_t^i = \sigma(W_{Ig^i} x_t + W_{Hg^i} h_{t-1} + b_{g^i}) \tag{4.5}$$

$$g_{\text{t}}^{\text{f}} = \sigma(W_{Ig^f} x_t + W_{Hg^f} h_{t-1} + b_{g^f}) \tag{4.6}$$

$$g_{\text{t}}^{\text{o}} = \sigma(W_{Ig^o} x_t + W_{Hg^o} h_{t-1} + b_{g^o}) \tag{4.7}$$

$$g_{\text{t}}^{\text{c}} = g \odot {}_{\text{t}}^{\text{i}}\tanh(W_{Ig^c} x_t + W_{Hg^c} h_{t-1} + b_{g^c}) + g_{\text{t}}^{\text{f}} \odot g_{\text{t-1}}^{\text{c}} \tag{4.8}$$

$$g_{\text{t}}^{\text{m}} = \frac{1}{H} \sum_{j}^{H} g_{\text{t}}^{\text{c,j}}, \quad v_t = \sqrt{\frac{1}{H} \sum_{j}^{H} (g_{\text{t}}^{\text{c,j}} - g_{\text{t}}^{\text{m}})^2} \tag{4.9}$$

$$h_t = g_{\text{t}}^{\text{o}} \odot \tanh(\frac{\gamma}{v_t} \odot (g_{\text{t}}^{\text{c}} - g_{\text{t}}^{\text{m}}) + \beta) \tag{4.10}$$

In this case, g_{t}^{c} and h_{t-1} represent the cell memory and the cell state from the previous time-step. Once again, x_t denotes the input, this time assumed as a human pose at time-step t. And as previously mentioned, $\sigma(\cdot)$ and the Hadamard $\odot$ product represent the element-wise sigmoid function and the element-wise product, H denotes the hidden units in the LSTM, W, γ and β are learned while γ and β have the same dimension of h_t. Contrary to the standard LSTM, the hidden state h_t is calculated by the normalization of g_{t}^{c}.

The Self-Attention Mechanism

A self-attention mechanism proposed by Zhouhan et al. [131] was implemented. Thus, In the context of human motion sequences, certain poses convey more information than others in an intuitive manner. As a result, the self-attention mechanism is employed to assign a score to each pose within the sequence. To be precise, the assumption is made that the sequence of states, denoted as $S = \{h_1, ..., h_t\}$ determined from a motion sequence X, which consists of n time-steps. Accordingly, the scoring function can be described as:

$$r = W_{s2}\tanh(W_{s1}S^T) \text{ and } a_i = -log(\frac{exp(r_i)}{\sum_j exp(r_{i,j})}) \tag{4.11}$$

where r has values in the range of [0,1] and is used to give different weights to a_i. Embedding a_i each time-step is weighted in the attention score $SC = r \cdot a$. Moreover, W_{s_1,s_2} are a weight matrix and a weight vector forming the attention module [88], [131], [132].

Implementation

As previously described in the Figure 4.2. The model consists of three main branches, each one consisting of an attention based, layer normalized BiLSTM or LNLSTM. the LNLSTM does a forward and backward pass through the whole given sequence, therefore results can not be inferred in real-time. Let $s_t = [s_{t,f}, s_{t,b}]$ such that $s_{t,f} = \overrightarrow{LNLSTM}(w_t, x_t)$ for $t \in [0, N]$ and $s_{t,b} = \overrightarrow{LNLSTM}(w_t, x_t)$ for $t \in [N, 0]$.

In n time steps of an input sequence X, $S(s_1, s_n)$ is calculated, where s_t is the concatenated output of the forward and backward passes of the LNLSTM with H (default at 128) hidden units. The BiLSTM is then linked to the dropout function and the batch normalisation function. The output is passed to an attention layer, which produces a fixed size output. After the attention layer, the network is terminated by the following sequence of structures: {fully connected(320), dropout, batch normalisation, fully connected(320), batch normalisation, fully connected (128), batch normalisation and l2 Norm}, where the dropout is limited to 0.5. Except for the last fully connected layer (FC), rectified linear units (ReLU) are applied to all FC layers. The self-attention mechanism used in this implementation is based on the approach described by Zhouhan et al [131]. For the scoring function in the attention mechanism, $W_{s_1} \in \mathbb{R}^{200\times10}$ and $W_{s_2} \in \mathbb{R}^{10\times1}$. Finally, random orthogonal matrices are used in this model to initialise all squared weight matrices, while the remaining matrices are initialised with a uniform distribution with a mean of zero and a standard deviation of 0.001, as per Coskun et al. [88]. The parameter γ is initialised with zeros and the parameter β is initialised with ones according to equation 4.10.

Training

For this step, the previous dataset from "The PROZIS Challenge" [125], [126] of several workout annotated exercises with extra annotations for the errors in video format was used. In previous work within the INPACT project and using this dataset, the virtual skeletal landmarks were extracted and fixed at the hip joint, labelled with different sections, i.e. up and down phases of a squat, and saved as multiple .JSON files. These files were then divided into two different categories for further processing (correctly performed exercises and exercises with execution errors). As the exercises varied in length, they were subsampled for length or, if the length of the sequence was below the desired threshold, padded or interpolated data was added. The data were split into training, validation and test sets, using a batch size of 32 samples (which proved, empirically, sufficient for our purposes).

The model used an Adagrad optimiser that minimised the binary cross-entropy function for binary classification with a starting learning rate of 0.001, and a scheduler that gradually reduced the learning rate by a factor of 0.8, with a patience of 7 epochs and a total of about 1000 sequences to train. Using an Nvidia RTX 2060 GPU, each epoch took about 4.2 seconds to train. Most training sessions were tested with a number of epochs between 300 and 1000, to test convergence. Most sessions reached the convergence plateau before the

500th epoch.

4.3 Motion Prediction

To further quantify the evaluation of the human movement in a rehabilitation setting, another framework was proposed, a network that could estimate or predict the next x frames, based on an average model, to then compare them to the ground truth, allowing the feedback to be at the joint level and not just an overall scoring function. To this effect, two graph based networks were chosen.

4.3.1 Space-Time-Separable Graph Convolutional Network

This graph-based approach involves encoding observed body joint coordinates from input images, followed by using a space-time representation to predict future joint coordinates. The encoding process is performed using the proposed Space-Time Separable Graph Convolutional Network (STS-GCN), which captures the interaction between body joints over time and focuses on space-time dynamics, as illustrated in Figure 4.3. The decoding of future coordinates is performed with a Temporal Convolutional Network (TCN) [25].

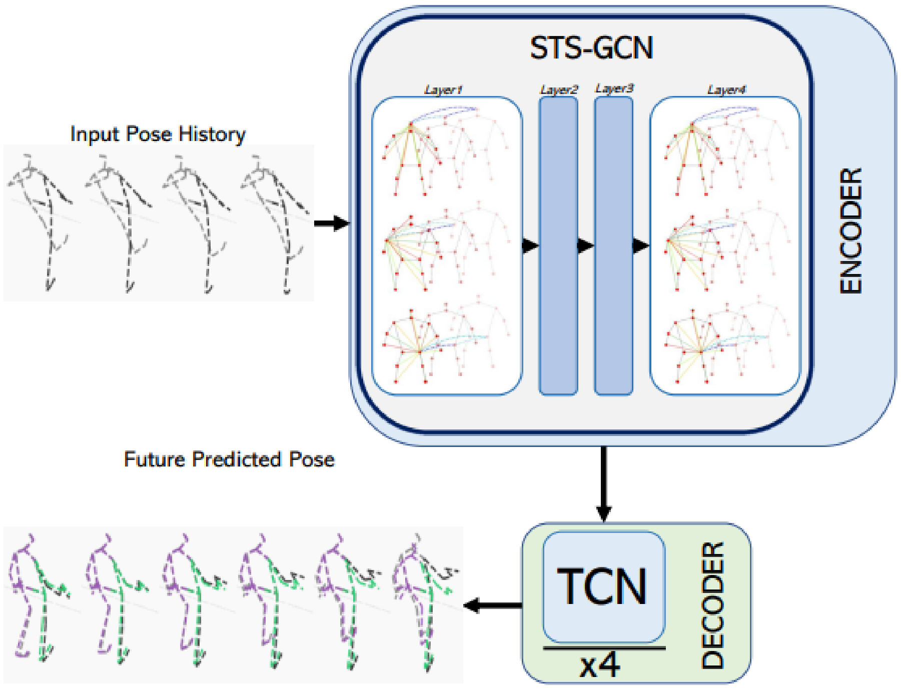


Figure 4.3: Pipeline of the STS-GCN according to Sofianos et al. [25]

Problem Formalization

Following the work of Sofianos et al. [25] the task involves analysing the body pose of an individual by examining the 3D coordinates or angles of their V joints over a span of T frames. The goal is to subsequently forecast the V body joints for the upcoming K frames.

Let the 3D joint vector $x_{x,k}$ represent the joint v at time-step k. The motion history of the poses denoted by the tensor, $\mathcal{X}_{in} = [X_1, X_2, ..., X_T]$ which can be used to build the matrices of the 3D coordinates, $X_i \in \mathbb{R}^{3\times V}$ for the time sequence $i = 1, ..., T$. The goal is to predict the next K $\mathcal{X}_{out} = [X_{T+1}, ..., X_{T+K}]$. A graph is created by encoding the motion history tensor, representing the interactions among all body joints throughout the observed frames. Accordingly, the graph can be denoted $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with TV nodes $i \in \mathcal{V}$, which represent the body joints across the whole time sequence. The edges $(i, j) \in \mathcal{E}$ are represented by a spatial-temporal adjacency matrix $A^{st} \in \mathbb{R}^{VT\times TV}$, connecting the interactions of the joints at all times [25].

Graph Convolutional Network

As aforementioned, spatio-temporal can be easily encoded by a Graph Convolutional Network $f(\mathcal{X}_{\rangle\backslash}, \mathcal{A}, \mathcal{W})$. The input layer to a convolutional layer l is the tensor $\mathcal{H}^l \in \mathbb{R}^{C^l\times V\times T}$ that encodes the V joints in T time-steps. C^l denotes the input dimensionality of H^l. In the first layer, $\mathcal{H}^L = \S_{in}$ and C^l=3. The graph convolutional layer l outputs $\mathcal{H}^{l+1} \in \mathbb{R}^{C^{l+1}\times V\times T}$ expressed by the following equation:

$$\mathcal{H}^{l+1} = \sigma(A^{st-l}\mathcal{H}^l W^l) \tag{4.12}$$

where $A^{st-l} \in \mathbb{R}^{VT\times TV}$ is the spacial-temporal adjacency matrix of l, and the trainable graph convolutional weights of l, $W^l \in \mathbb{R}^{C^l\times C^{l+1}}$, projecting each graph node C^l to C^{l+1} and σ denotes the activation function.

Spatio-Temporal Separable Convolutions

The STS-GCN as proposed by [25] draws inspiration from the interplay between the temporal progression and spatial connections of joints, emphasizing the significance of joint-joint and time-time interactions. The dynamics of human pose involve three distinct types of interactions: joint-joint, time-time, and joint-time. While STS-GCN accommodates all three interactions, it restricts the joint-time cross-talk to a minimum. To capture the interplay of joints over time, the model utilizes a single spatio-temporal encoding GCN that relates the three types of relations. The bottlenecking of space-time cross-talk is achieved by decomposing the space-time adjacency matrix into separate spatial and temporal adjacency matrices $A^{st} = A^s A^t$. thus convolutional layer l can be described as:

$$\mathcal{H}^{l+1} = \sigma(A^{s-l} A^{t-l} \mathcal{H}^l W^l) \tag{4.13}$$

Here, the same notation of the equation 4.12, except that $A^{s-l}A^{t-l}$ is the factorised adjacency matrix of the plane l. The interaction of the joints is determined by the adjacency matrix $A^s \in \mathbb{R}^{V\times V}$, which represents the complete relations between the joints by trainable matrices of size $V \times V$ at each time point (with a total of T such matrices). Similarly, the temporal relations are defined by the adjacency matrix $A^t \in \mathbb{R}^{T\times T}$, which consists of trainable $T \times T$ matrices representing the complete temporal relations for each of the V Joints.

Equation 4.13 illustrates a single layer of the (GCN) that captures the interplay between spatial and temporal aspects of body dynamics. By employing a factorized space-time matrix, the model effectively limits the exchange of information across space and time, resulting in a reduction of model parameters. As a result, there is a significant improvement in the forecasting performance.

4.3.2 Spatio-Temporal Separable Convolutions and Self-Attention

Rewriting the GCN equation as shown in 4.12 with the Einstein summation, omitting the layer l, with the projection matrix W and the non-linearity of the activation function for better clarity:

$$A^{st}\mathcal{H} = \sum_{vm} A^{\mathrm{st}}_{\mathrm{wkvm}}\ \mathcal{H}_{vmc} \tag{4.14}$$

where $A^{st} \in \mathbb{R}^{VT\times TV}$ and $\mathcal{H} \in \mathbb{R}^{C\times V\times T}$, and indexing spatial joints as $v, w = 1, ..., V$ and time-steps $m, k = 1, ..., T$. Rewriting the corresponding part of the space-time separable convolution (STS-GCN) equation 4.13, with the Einstein summation, while, once again, omitting W and the activation function, σ:

$$A^s(A^t\mathcal{H}) = \sum_v A^{\mathrm{s}}_{\mathrm{wkv}}(\sum_m A^{\mathrm{t}}_{\mathrm{kvm}}\mathcal{H}_{vcm}) = \sum_v A^{\mathrm{t}}_{\mathrm{wkv}}\mathcal{H}^{\mathrm{t}}_{\mathrm{kvc}} \tag{4.15}$$

the notation for equation 4.13 applies for equation 4.15, where there are indicated indexes for $A^s \in \mathbb{R}^{V\times V}$, for each of the time-steps and $A^t \in \mathbb{R}^{T\times T}$, for each of the joints as $v, w = 1, ..., V$ for the spatial joints and $m, k = 1, ..., T$ for the times-steps [25].

Decoding the future coordinates

The convolutional layers, which are responsible for the temporal dimension, take the encoded observed body dynamics and use them to estimate the future 3D coordinates or angles of the body joints. In this process, the observed images are mapped to the future horizon and the estimates are refined through a multi-layer architecture.

Training

For the training of this architecture, the loss function minimized the error in relation to the ground truth through Mean Per Joint Position Error (MPJPE), denoted as:

$$L_{MPJPE} = \frac{1}{V(T+K)} \sum_{k=1}^{T+K} \sum_{v=1}^{V} ||\hat{x}_{vk} - x_{vk}||_2 \tag{4.16}$$

with $\hat{x}_{vk} \in \mathbb{R}^3$ denoting the predicted coordinates of the joint v in the time-step k and $x_{vk} \in \mathbb{R}^3$ is the corresponding ground truth [25].

For this method, the Human3.6M dataset [1] was used, both after Jain et al.'s [26] conversion of 3D Euler angles to exponential map and using raw Cartesian coordinates, with the skeleton's articulations being capped at maximum of 22 joints.

As Sofianos et al. [25] stated, the encoding is built by 4 layers of STS-GCN, each layer has a batch normalization function and residual connections. An Adam optimizer was used with a learning rate of 0.001 decaying by a factor of 0.1 with patience of 5 epochs and a batch sizer of 256. Training 30 epochs on an Nvidia RTX 2060 took between 20–30 minutes.

4.3.3 Spatial-Temporal Anchor-based Sampling

This method is an improved version of the earlier STS-GCN by Sofianos et al. [25] and other work by Martinez et al. and Jain et al. [23], [26], by using deep generative models to produce a manifold prediction. Multi-level Spatial-Temporal AnchoR-based Sampling (STARS) as in Figure 4.4, developed by Sirui et al. [102] and whose main insight is that future movements are not completely random or unrelated. They have certain predictable characteristics that are influenced by physical laws and the limitations of the human body. They also tend to follow patterns that have been created by previous movements. To illustrate, some imminent movements may have expected changes in speed or direction determined by common factors, while their exact magnitude may vary randomly.

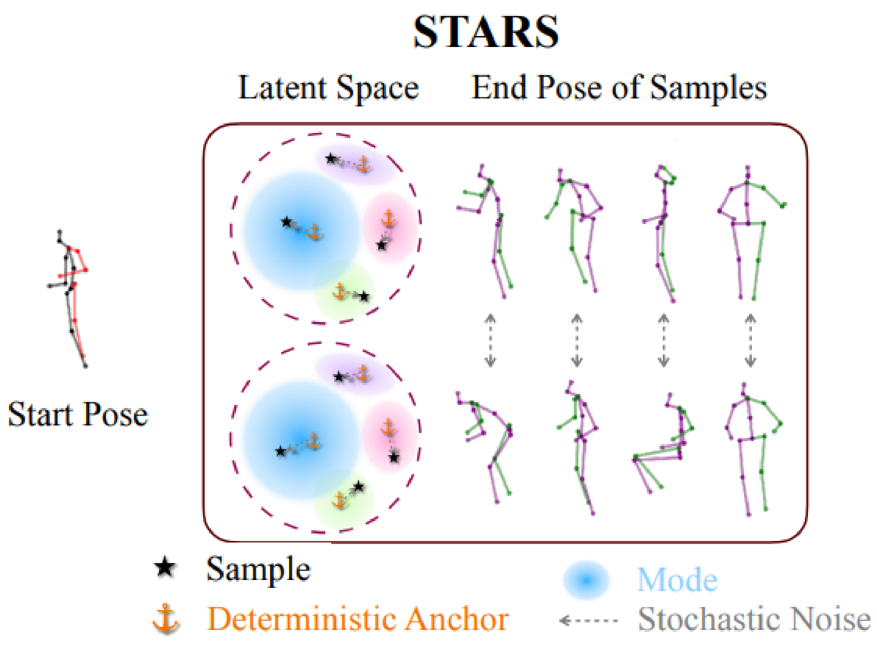


Figure 4.4: Spatial-Temporal AnchoR-based Sampling [102]

Problem Formalization

According to the notation of Sirui et al. [102], an input sequence T_h with the 3D coordinates of V joints describing each pose, $X[x_1, ..., x_{T_h}]^T$, $x_i \; in \mathbb{R}^{V \times C^0}$, where $C^0 3$ denotes the 3 dimensions, K Output sequences of length T_p, denoted $\hat{Y}_1, ..., \hat{Y}_K$ and the single ground truth, denoted Y. The focus is on predicting a sequence K that is as close as possible to the ground truth, and these K sequences are as diverse as possible and represent a realistic future movement [102].

As the Figure 4.5 shows, this is true for different types of generative models: (a) In a conventional generative model, stochastic noise is the only factor that is optimised. (b) In a generative model with a deterministic anchor process, an anchor with Gaussian noise is used as prediction. (c) Spatio-temporal compositional anchors correspond to predictions, considering any combined pair of spatial and temporal anchors. (d) Multi-level spatio-temporal anchors encode multiscale modes by combining anchors at different levels [102].

Deep Generative Models

Numerous studies have looked in depth at generating multiple hypotheses using deep generative models. These models typically involve learning a parametric probability distribution function, either explicitly or implicitly, to predict future movements. However, the stochastic nature of these models can lead to variations in the magnitude of the hypotheses generated.

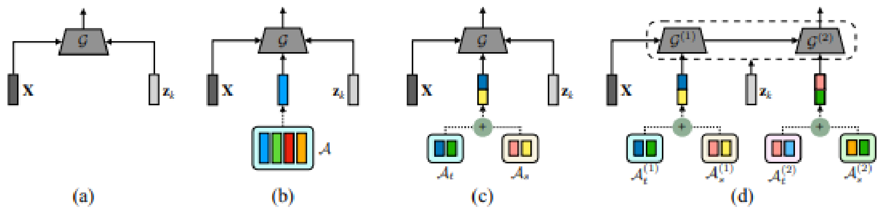


Figure 4.5: The optimization process involves the joint optimization of anchors and network parameters [102]

Let $p(Y|X)$ describe the probability distribution function of the movement Y, conditioned on the past movement X, with the latent variable $z \in \mathcal{Z}$, thus the distribution can be reparameterised as $p(Y|X) \int p(Y|X,z)p(z)\mathrm{d}z$, where $p(z)$ is a prioritised normal distribution. To generate $\hat{Y}$), z is drawn from $p(z)$ and then mapped using a deterministic generator $\mathcal{G} : \mathcal{Z} \times \mathcal{X} \rightarrow \mathcal{Y}$ as [102] (Figure 4.5 a)):

$$z\ p(z),\ \hat{Y} = \mathcal{G}(z, X) \tag{4.17}$$

where $\mathcal{G}$ represents a deep neural network parameterized by θ, where it is attempted to make the distribution $p_\theta(\hat{Y}|X)$ (from $\mathcal{G}$) as close as possible to $p(Y|X)$. In order to produce a variety of motion predictions, conventional methods typically start by randomly selecting a group of latent codes $Z = \{z_1, ..., z_K\}$ from a prior distribution $p(z)$. However, while generative models have the potential to encompass various modes, they do not offer a definite assurance of accurately capturing all modes [102], [133], [134].

Anchor-Based Sampling

To cope with the problem stated in the last section, the author decomposes the code within the latent space of the generative model into two parts: a stochastic component sampled from $p(z)$, and a deterministic component represented by a collection of K learnable parameters known as anchors $a = \{a_k\}_{\mathrm{k}=1}^{\mathrm{K}}$, taking into consideration some previous assumptions:

- The deterministic aspect refers to the presence of correlated or shareable changes in velocity, direction, movement patterns, etc., that naturally emerge across various actions performed by different subjects and can be directly acquired from data.

- The stochastic aspect pertains to the random magnitude of the changes that occur when a subject carries out an action.

Thus, the intuition behind deterministic anchors is to detect a wide range of patterns, and this is accomplished by employing a meticulously crafted optimization process. Additionally, stochastic noise is utilized to add further details to the variations in movement within specific patterns. The latent code disentanglement can be denoted as a new multi-modal distribution:

$$p_\theta(\hat{Y}|X,\mathcal{A}) = \frac{1}{K}\int p_\theta(\hat{Y}|X,z,a_k)p(z)\mathrm{d}z \tag{4.18}$$

Accordingly, the selected k-th learned anchor $a_k \in \mathcal{A}$, with randomly sampled noise $z \in Z$, can generate a prediction $\hat{Y}_k$:

$$z\ p(z), \hat{Y}_k = \mathcal{G}(a_k, z, X) \tag{4.19}$$

a total of K predictions can be made if an anchor is used only once, even though the anchors are not limited to being used once. To incorporate the anchors into the network, it is more effective if simple additions are made between anchors and latent features, Figure 4.6 b).

Spatial-Temporal Anchors

As mentioned in the sections on graph networks, the range of potential future movements can be roughly divided into two distinct forms: spatial variation and temporal variation, which are relatively separate. This observation suggests a viable approach to further divide K anchors into two types of adaptable codes: spatial anchors $A_s\{a^{\mathrm{s}}_{\mathrm{i}}\}^{\mathrm{Ks}}_{\mathrm{i1}}$and temporal anchors $A_t\{a^{\mathrm{t}}_{\mathrm{j}}\}^{\mathrm{Kt}}_{\mathrm{j1}}$. The usual notation of $KK_s \times K_t$ can be used here. Thus, with this decomposition, $K_s \times K_t$ compositional anchors can be obtained. The frequency variation of future movement sequences is actually controlled by the temporal anchors, since they represent temporal features in the frequency domain. Spatial anchors, on the other hand, are conceptually identical in the temporal dimension, but describe the spatial variation of movement and control the general trends and directions of movement. Conversely, temporal anchors remain constant in the spatial dimension but differ in the temporal dimension, resulting in frequency differences that affect the speed of the movement.

Thus, to generate $\hat{Y}_k$ (Figure 4.5 c)), z must be sampled and the i-th and j-th spatial ($a^{\mathrm{s}}_{\mathrm{i}}$) and temporal ($a^{\mathrm{t}}_{\mathrm{j}}$) anchors chosen:

$$z\ p(z),\ \hat{Y}_k = \mathcal{G}(a^{\mathrm{s}}_{\mathrm{i}} + a^{\mathrm{t}}_{\mathrm{j}}, z, X) \tag{4.20}$$

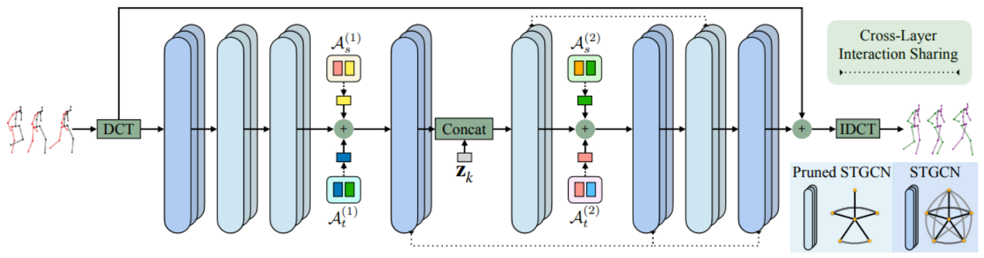


Figure 4.6: Combination of ST anchors, sampled noise with the backing STGCN [102]

Again, $a_i^s a_j^t$ stands for the compositional anchor s-t. In addition, the spatio-temporal anchors allow movement control to be adjusted via spatial and temporal variations. To illustrate, if you keep the spatial anchors constant and adjust or interpolate the temporal anchors, it is possible to create future movements that have comparable patterns [102].

Multi-Level S-T Anchors

To capture multiscale modes of future movement, Sirui et al. [102] proposed a multilevel mechanism to extend the spatio-temporal anchors. This is illustrated in Figure 4.5 (d), which shows a double-storey or two-stage design. The author introduces two different sets of s-t anchors $\{A_t^1, A_t^1\}$ and $\{A_t^2, A_t^2\}$ and associates them with two different network parts $\mathcal{G}^1, \mathcal{G}^2$. Assuming that (i, j) are the s-t indices corresponding to the 1D index k, the sequence $\hat{Y}_k$ can be generated by a two-step process as:

$$z\ p(z),\ \hat{Y}_k = \mathcal{G}^2(a_i^{s2} + a_j^{t2}; z, \mathcal{G}^1(a_i^{s1} + a_j^{t1}; X)) \tag{4.21}$$

where $a_i^{s1} \in A_s^1$, $a_i^{t1} \in A_t^1$, $a_i^{s2} \in A_s^2$, $a_i^{t2} \in A_t^2$. Anchors can thus be used at several levels and serve as a principled approach to incorporate more complicated assumptions about impending movements.

During the training process, the model uses each spatio-temporal anchor in an explicit way to generate K future movements for each past movement sequence. The loss functions essentially follow the approach outlined in [25], [135], which can be divided into three main types: (1) reconstruction losses, which optimise the best predictions given different definitions among the K generated motions, ensuring that the anchors match their respective closest modes; (2) a diversity-promoting loss, which explicitly promotes different pairwise distances in the predictions and prevents the anchors from collapsing into a single mode; and (3) motion constraint losses, which promote the generation of realistic output motions. The anchors are learned directly from the data using gradient descent. During the forward pass, each anchor a_i is added as an additional input to the network, resulting in a total of K outputs. During the backward pass, each anchor is independently optimised based on its corresponding outputs and losses, while the backbone network is updated based on the combined losses of all outputs.

Interaction-Enhanced Spatial-Temporal Graph Convolutional Network

Let the Discrete Cosine Transform (DCT) $\hat{X}_{1:T_hT_p}$ be a past motion where each pose has V joints and a predefined mask M, and the base $C \in \mathbb{R}^{M\times(T_hT_p)}$ is a past pose $X_{1:T_h}$, where the last pose is replicated to be $X_{1:T_hT_p}[x_1, ..., x_{T_h}, x_{T_h}, x_{T_h}, ..., x_{T_h}]^T$. The formulation of $\hat{X}CX_{1:T_hT_p}\ in\mathbb{R}^{M\times V\times C^0}$ in the 0-th layer and the s-t features of any layer l as $(\mathcal{V}^l, \mathcal{E}^l)$ with $M \times V$ nodes. The node i is given by the 2D index (f_i, v_i) for the joint v_i and the frequency component of the DCT f_i. The edge $(i, j) \in \mathcal{E}^l$ is the link between node i and node j and is represented by the learnable adjacency matrix $Adj^l Adj^{\mathrm{l}}_{\mathrm{s}}\ Ads^{\mathrm{l}}_{\mathrm{f}}\ \in \mathbb{R}^{MV_MV}$. $Adj^{\mathrm{l}}_{\mathrm{s}}$, connects the nodes with the same frequency and $Adj^{\mathrm{l}}_{\mathrm{f}}$ is responsible for the interactions between the nodes representing the same joint [102].

As in the previous method, the s-t graph could be encoded via graph convolution network, given a set or trainable weights $W^l \in \mathbb{R}^{MV\times MV}$ and the activation function $\sigma(\cdot)$, the STGCN layer projects the input dimensions from C^l to C^{l+1} by:

$$\mathcal{H}^{\mathrm{l+1}}_{\mathrm{k}} = \sigma(Adj^l\mathcal{H}^{\mathrm{l}}_{\mathrm{k}}W^l) = \sigma(Adj^{\mathrm{l}}_{\mathrm{s}}Adj^{\mathrm{l}}_{\mathrm{f}}\mathcal{H}^{\mathrm{l}}_{\mathrm{k}}W^l) \tag{4.22}$$

where, once again $\mathcal{H}^{\mathrm{l}}_{\mathrm{k}}\ \in\ \mathbb{R}^{MV\times C^l}$ denotes the latent feature or hidden state of the prediction $\hat{Y}_k$ at the l-th layer. As in the previous method, the whole of the network consists of multiple GCNs. After the prediction of the DCT coefficients $\tilde{Y}_k\ \in\ \mathbb{R}^{M\times V\times C^L}$ reshaped from $\mathcal{H}^{\mathrm{L}}_{\mathrm{k}}$ and $C^L = 3$, we can recover $\hat{Y}_k$ via the inverse DCT as:

$$\hat{Y}_k = (C^T\tilde{Y}_k)_{T_h+1:T_h+T_p} \tag{4.23}$$

where the last T_p time-steps are recovered, representing the future, predicted poses[102].

Cross-Layer Sharing and Spatial Interaction Pruning

Sirui et al. also found that sharing adjacency matrices at a step of one layer is quite effective, i.e. Adj^4_{s} and $Adj^6_{\mathrm{s}}Adj^8_{\mathrm{s}}$ and $Adj^5_{\mathrm{s}}Adj^7_{\mathrm{s}}$, as can be seen in the Figure 4.6. To highlight the physical relationships and constraints between spatial joints, the spatial connections $\hat{Adj}^{\mathrm{l}}_{\mathrm{s}}M_s \odot Adj^{\mathrm{l}}_{\mathrm{s}}$ within each graph layer l are pruned by applying a predefined mask M_s using the element usage product $\odot$. The author thus proposes:

$$M_s[i][j] = \begin{cases} 1 & v_i \text{ and } v_j \text{ are physically connected, } f_i = f_j \\ 1 & v_i \text{ and } v_j \text{ are mirror connected, } f_i = f_j \\ 0 & \text{otherwise} \end{cases} \tag{4.24}$$

In conclusion, the architecture comprises four STGCNs in their original form, without any spatial pruning, along with four Pruned STGCNs.

4.3.4 Training

As aforementioned, the loss functions can be categorized into three groups:

- The reconstruction losses, which encompass reconstruction error and multi-modal reconstruction error.
- The diversity-enhancing losses (Multi-Modal).
- The motion constraint losses, which consist of history reconstruction error, pose prior, limb loss, and angle loss.

Reconstruction error loss, with the aim to minimize the reconstruction error by promoting the prediction that closely aligns with the ground truth. This, in turn, encourages the corresponding anchor to capture a specific pattern or mode is given by the following equations which share the same notation as the previous subsections:

$$L_r = \min_k ||\hat{Y}_k - Y||^2 \tag{4.25}$$

The multi-modal reconstruction error works by encouraging predictions to encompass multimodal ground truth, there is a promotion of anchors to capture a wider range of modes.

$$L_{mm} = \frac{1}{N} \sum_{n=1}^{N} \min_k ||\hat{Y}_k - Y_n||^2 \tag{4.26}$$

The multimodal ground truth Y_n represents the possible future movements in several modes. The discontinuity between prediction and history is mitigated by a close fit of the reconstructed historical movement $\hat{X}_k$ to the past sequence of the ground truth X, which is achieved by using the inverse DCT. The error in the reconstruction of history can thus be defined as follows

$$L_h = \frac{1}{K} \sum_{k=1}^{K} ||\hat{X}_k - X||^2 \tag{4.27}$$

The diversity-promoting loss function is designed to prevent the anchors from converging to the same point by actively promoting differences between pairwise predictions, defined as:

$$l_d = \frac{2}{K(K-1)} \sum_{j=1}^{K} \sum_{k=j+1}^{K} exp(-\frac{||\hat{Y}_j - \hat{Y}_k||_1}{\alpha}) \tag{4.28}$$

The use of a pre-trained normalisation flow p_nf to evaluate the probability of the pose already present enables the measurement of generated human poses $\hat{Y}_k$. Using this module ensures that the generated poses have a considerable probability within the p_nf.

$$L_{nf} = -\sum_{k=1}^{K} \log_{p_nf}(\hat{Y}_k) \tag{4.29}$$

Limb loss is addressed by ensuring that the length of the limb (bone) remains consistent with the ground truth. The bone length is determined as the distance between two joints that are physically connected, and the vector $\hat{L}_k$ encompasses the bone lengths of all poses within $\hat{X}_k$. Thus, the limb loss can be given by:

$$L_l = \frac{1}{K} \sum_{k=1}^{K} ||\hat{L}_k - L||^2 \tag{4.30}$$

Finally the training parameters were kept as Sirui et al. [102] stated. 8 layers of STGCN starting with 3 channels, then 128,64,128,64,128,64,128 and once again 3 channels. The learning rate was set at 0.001 and decayed after 100 epochs, as:

$$lr = 0.001 \times (1.0 - (\frac{max(0, epoch - 100)}{400})) \tag{4.31}$$

The optimizer used to train the model was the Adam function and capped the number of possible K predictions to 1. Multiple cases were trained with 10 frames of context and between 5 and 50 frames of predicted movement.

5 Results and Discussion

In this chapter, we will present and discuss the results of different performance evaluation tasks. As mentioned earlier, two different tasks, skeleton-based action recognition and motion prediction, are evaluated using three models, one of which focuses on action recognition and two competing models are used for motion prediction. Quantitative and qualitative performance metrics are extracted for these three models, and conclusions are drawn about their feasibility in a real-time scenario.

5.1 Metrics and Evaluation

In order to evaluate the proposed methods, several metrics were established. These were separated by task. Since the action recognition task is a classification task and the dataset is unbalanced, some metrics need to be carefully selected to best represent the overall performance of the method. For this purpose, the false positive rate (FPR), the F1 score and the mean class accuracy were chosen.

Let TP denote the true positive cases, TN the true negatives, FP the false positives and consequently FN the false negatives of the i-th class out of a total of N classes, $i \in N$:

$$mCA = \frac{1}{N}\sum_{i=1}^{N}\frac{TP+TN}{TP+TN+FP+FN} \tag{5.1}$$

The mean-class accuracy is used because the distribution is not balanced. In an ideal scenario, the distributions of samples between subsamples (classes) would be uniform, but because each set has different samples and a varying number of them, the accuracy would undoubtedly change as well. So, to remediate this problem, mean-class accuracy was introduced to achieve more realistic results.

The F1-score is a better way of measuring unbalanced data than the standard accuracy. The F1 score has two parts, the precision and the recall, it is used to measure the balance between precision and recall. It considers both the false positives and false negatives in determining the overall performance of a model. The F1 score is calculated as the harmonic mean $(\frac{1}{\frac{1}{2}})$ of precision and recall, providing a single value that represents the model's accuracy in classifying positive instances while minimizing incorrect classifications.

$$Precision = \frac{TP}{TP + FP} \tag{5.2}$$

$$Recall = \frac{TP}{TP + FN} \tag{5.3}$$

$$F1 = \frac{TP}{TP + \frac{1}{2}} = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{5.4}$$

Normalised mutual information (NMI) is a metric for measuring similarity or agreement between two groups of labels or clusters. It is commonly used in unsupervised learning tasks, such as clustering or community detection.

NMI calculates the mutual information between the two sets of labels and then normalises it to obtain a value between 0 and 1. A higher NMI value indicates greater similarity or agreement between the label sets, while a lower value indicates less similarity or agreement.

$$NMI(Y, C) = \frac{2 \times I(Y; C)}{[H(Y) + H(C)]} \tag{5.5}$$

where Y is the class labels, C the cluster labels, the function $H(\cdot)$ is the entropy, and $I(Y; C)$ is the mutual information of $Y \cap C$.

For the action recognition model, the final metric used to measure the overall performance of the model is the False Positive Rate as proposed by Coskun et al. [88]. The false Positive Rate (FPR) is a metric used to evaluate the performance of a binary classification model. It measures the proportion of negative samples that are falsely classified as positive. In other words, it represents the rate at which the model mistakenly identifies a negative sample as positive. The false positive rate is calculated by dividing the number of false positive predictions by the sum of true negative predictions and false positive predictions. It is often used in conjunction with other evaluation metrics, such as the true positive prediction rate (recall) or the false negative prediction rate (FNR), to provide a comprehensive understanding of the performance of the model.

$$FPR = \frac{FP}{FP + TN} \tag{5.6}$$

For the second part of the proposed method, a few methods were employed based on loss functions. The Mean Per Joint Position Error MPJPE, equation 4.16, is reported in millimetres.

5.2 Evaluation of the Baseline Methods

In this section, there will be a quantitative analysis between the chosen methods and other baselines existing in the literature.

5.2.1 Evaluation of Coskun's Attentive LSTM

For the quantitative assessment of the metric, the dataset Human3.6M [1] was used for comparison with the baseline. In the Results section, other metrics are analysed using the PROZIS dataset [125], [126].

When using the Human3.6M dataset, all actions and sub-actions from all subjects were used. The model was trained in 8 categories and tested in 7. The CMU [136] dataset was also used, with six joints excluded, and the sampling rate reduced from 120Hz to 30Hz. Of the 38 categories, 19 were selected for testing and 19 for training. To avoid gimbal locking, the data were processed into an exponential map [88], [137]. A series of FPR tests for 90%, 80% and 70% true positive rates were made, according to the literature, these values allow for a better visualisation of the performance of the classificator, i.e. if the values at 90%, 80% and 70% are relatively close to each other and zero, the classificator is quite good [88].

	CMU			H36M		
	FPR-90	FPR-80	FPR-70	FPR-90	FPR-80	FPR-70
DTW[138]	47.48	42.92	37.62	49.64	47.96	44.38
MDDTW[139]	44.60	39.07	34.04	49.72	45.87	44.51
CTW[140]	46.02	40.96	39.11	47.63	43.10	42.18
GDTW[141]	45.61	39.95	35.24	46.06	42.72	40.04
DCTW[142]	40.56	38.83	26.95	41.39	39.18	36.71
Triplet[143]	39.72	33.82	28.77	42.78	40.11	36.01
Triplet + GOR[144]	40.32	33.97	27.78	42.03	37.61	33.95
N_Pair[145]	40.11	32.35	26.16	40.46	39.56	36.52
MMD-NCA[88]	**32.66**	**25.66**	**20.29**	**38.42**	**36.54**	**33.13**

Table 5.1: Comparison of the FPR

If the true-positive rate (TPR) is set to $X\%$, this means that the model is expected to correctly identify X% of the positive instances.

However, the FPR is not determined by the TPR alone. It depends on the specific threshold or decision limit chosen by the model for classifying instances as positive or negative. Different thresholds will result in different FPR values even if the TPR is fixed.

To determine the FPR corresponding to a specific TPR, additional information must be calculated, such as the specific threshold or Receiver Operating Characteristic (ROC) curve for the model.

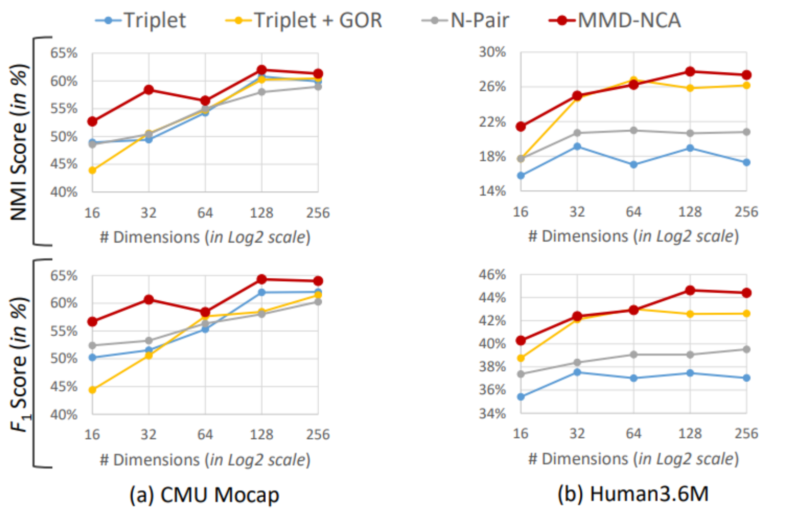


Figure 5.1: The F1 scores and Normalized Mutual Information of 3 of the best baselines and Coskun et al.'s method MMD-NCA [88]

It can be empirically deduced that in NMI and in F1 score, as per Figure 5.1, Coskun's approach works best results when compared to differing embedding sizes.

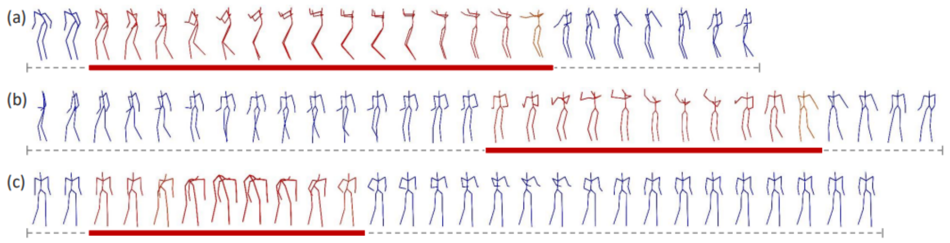


Figure 5.2: Attention mechanism in play [88]

The aim of the self-observation mechanism is to draw attention to the poses that contain the most informative details in terms of the semantics of the movement process. Consequently, the attentional mechanism is expected to prioritise the descriptive poses within the movement, allowing the model to capture more informative embeddings. Based on the peaks of A, which is composed of a_i from the equation 4.12.

Figure 5.2 models a basketball sequence. Regardless of the differences in the length of the movement, the model's attention is focused on the moment when the player throws the ball, as this contains the most informative aspect of the movement. Similarly, in the bending movement, the model focuses on the specific sections of the movement. This illustration thus shows the successful use of the mechanism of self-attention to highlight the most informative sections of the sequence.

5.2.2 Evaluation of STARS and STS-GCN

For this methodology, the two considered will be compared quantitatively and qualitatively. The comparisons will focus on the MPJPE error, on models trained on exponential map form. It should be noted that the model STARS, like the author's implementation [102], does not include a mode for 3D XYZ Cartesian joints. Further information on the results of the model STS-GCN, trained on 3D XYZ Cartesian joints, is also provided.

The dataset used was Human3.6M [1] with all 15 different actions trained for 5 subjects, using one subject as a test set and another as a validation set. Each pose is represented by 22 joints in an exponential map format and a 3D Cartesian format [25], [106].

All algorithms require 10 frames (400 msec) as input, although results are also presented for a higher number of input frames. The algorithms then generate predictions for future poses, ranging from 2 to 10 frames (80-400 msec) in the case of the short-term predictions and 14 to 25 frames (560-1000 msec) in the case of the long-term scenario [25], [106].

In addition to the two methods that are the focus of this paper, the comparison also includes ConvSeq2Seq [146], which uses convolutional layers to encode the long- and short-term histories separately. Also considered is LTD-X-Y [106], which incorporates a DCT to encode the sequence frequency before a GCN (X and Y represent the number of observed and predicted frames) [25], [102]. The results are as follows

	Walking				Eating				Smoking				Discussion			
	560	720	880	1000	560	720	880	1000	560	720	880	1000	560	720	880	1000
ConvSeq2Seq	72.2	77.2	80.9	82.3	61.3	72.8	81.8	87.1	60.0	69.4	77.2	81.7	98.1	112.9	123.0	129.3
LTD-50-25	50.7	54.4	57.4	60.3	51.5	62.6	71.3	75.8	50.5	59.3	67.1	72.1	88.9	103.9	113.6	118.5
STS-GCN	60.3	64.6	65.9	70.02	57.2	68.3	75.5	82.6	54.2	63.8	70.8	76.1	91.8	105.2	113.8	118.9
STARS	**49.3**	**53.5**	**57.4**	**61.1**	**50.2**	**61.1**	**69.1**	**74.1**	**44.2**	**51.8**	**59.0**	**64.3**	**74.0**	**85.1**	**94.1**	**100.4**

	Directions				Greeting				Phoning				Posing				Purchases				Sitting			
	560	720	880	1000	560	720	880	1000	560	720	880	1000	560	720	880	1000	560	720	880	1000	560	720	880	1000
ConvSeq2Seq	86.6	99.8	109.9	115.8	116.9	130.7	142.7	147.3	77.1	92.1	105.5	114.0	122.5	148.8	171.8	187.4	111.3	129.1	143.1	151.5	82.4	98.8	112.4	120.7
LTD-50-25	**74.2**	**88.1**	**99.4**	**105.5**	104.8	119.7	132.1	136.8	68.8	83.6	96.8	105.1	110.2	137.8	160.8	174.8	**99.2**	**114.9**	**127.1**	**134.9**	79.2	96.2	110.3	118.7
STS-GCN	75.8	92.9	102.2	109.6	111.2	122.4	131.8	136.1	72.5	87.9	99.7	108.3	115.8	142.4	161.7	178.4	104.6	119.4	132.7	141.0	82.0	97.6	110.9	121.4
STARS	76.4	91.6	102.4	107.6	**101.2**	**116.5**	**129.5**	**135.8**	**67.2**	**82.1**	**95.1**	**103.8**	**79.7**	**98.2**	**113.9**	**129.3**	100.3	117.5	133.1	139.4	**77.5**	**95.0**	**109.1**	**117.6**

	Sitting Down				Photo				Waiting				Dog Walking				Walking Together				**Average**			
	560	720	880	1000	560	720	880	1000	560	720	880	1000	560	720	880	1000	560	720	880	1000	560	720	880	1000
ConvSeq2Seq	106.5	125.1	139.8	150.3	84.4	102.4	117.8	128..1	87.3	100.3	110.7	117.7	122.4	133.8	151.1	162.4	72.0	77.7	82.9	87.4	90.7	104.7	116.7	124.2
LTD-50-25	**100.2**	**118.2**	**133.1**	**143.8**	**75.3**	**93.5**	**108.4**	**118.8**	77.2	90.4	101.1	108.3	**107.8**	**120.3**	**136.3**	**146.4**	56.0	60.3	63.1	**65.7**	79.6	93.6	105.2	112.4
STS-GCN	104.1	121.4	137.6	148.4	81.2	99.6	111.3	80.3	95.0	95.0	105.9	113.6	119.0	129.0	143.9	151.5	61.9	65.4	69.1	72.5	85.0	98.3	108.9	117.0
STARS	100.9	120.7	136.6	146.8	78.1	96.9	112.1	122.2	**71.7**	**85.0**	**96.0**	**103.6**	111.9	126.3	142.9	153.1	**54.1**	**59.5**	**63.5**	67.5	**75.8**	**89.3**	**100.8**	**108.4**

Table 5.2: Comparative analysis of the obtained results.

It is possible to see that in this segment the STARS' IE-STGCN for long term predictions does, indeed, achieve better MPJPE error values than the STS-GCN and even LTD with 50 context frames. Unfortunately, due to the LTD's fixed size, it does not allow great versatility. Even though the IE-STGCN achieves better results, these are not significative, achieving in a long term prediction around 10 mm better prediction accuracy. For short term, prediction, the average is even less noticeable. The table below shows that the difference of the average for prediction of about 10 frames is no larger than 8 mm.

	Short-term prediction				Long-term prediction			
	80	160	320	400	560	720	880	1000
ConvSeq2Seq	16.6	33.3	53.9	61.2	90.7	104.7	116.7	124.2
LTD-50-25	11.2	23.4	47.9	58.9	79.6	93.6	105.2	112.4
STS-GCN	13.5	27.7	54.4	65.8	85.0	98.3	108.9	117.0
STARS	**10.0**	**21.8**	**45.7**	**56.9**	**75.8**	**89.3**	**100.8**	**109.4**

Table 5.3: Average Comparison between multiple methods.

Another important note is the prediction using the standard 3D XYZ Cartesian point coordinate system. The prediction using 3D XYZ points is not as great as with the exponential map transformation, but the data requires less pre-processing and post-processing in the form of the introduction of inverse kinematics and forward kinematics functions. According to [25], using the 3D XYZ Cartesian coordinates reduces the performance of the model by 23% for short-term prediction and 34% for long-term prediction, which is sufficient for the purposes of this thesis.

5.3 Results

For the results, the focus will be on the qualitative assessment of the overall results and the provided feedback to the end-user. The architecture's successes and shortcomings. Firstly, for the action recognition purposes, there is a need to explain the PROZIS dataset [125], [126].

5.3.1 The PROZIS Challenge Dataset

This dataset is a custom, non-open source dataset containing videos of different annotated exercises (squats, burpees, sit-ups, push-ups and jumping jacks) from different participants of varying length (and unknown frame rate) and with added annotated execution errors [125], [126]. The capture and processing of this dataset was part of a project of the ISR (Institute of Systems and Robotics) of the Electrical and Computer Engineering Department of the University of Coimbra, named "The PROZIS Challenge" [125], [126]. Due to previous work related to the INPACT project, which aimed to transform this dataset from image based to skeleton based data, proved itself to be somewhat difficult. Some exercises were recorded in-house with several people passing by, in the background. This led to fragments in the extraction of the skeletal landmarks, which eventually delayed the correct extraction of the landmarks. Therefore, only a limited number of joint sequences were available for this project. These landmarks were stored in a .json file with the appropriate annotations for the beginning and end of a repetition, if the repetition had an execution error, and the skeletal embeddings. In the end, the squat exercise had the most samples with almost 950 sequences, the other exercises varied between about 300 and 70. It is also worth noting that there is a clear discrepancy between the number of well executed and the not well executed tasks.

5.3.2 Action Recognition

When it comes to action recognition, or the classification of exercises that are performed well and those that are not, the model had some difficulties with the PROZIS dataset [125], [126]. In the first place, the differences between well-executed exercises and not-well-executed exercises are very minute. Thus, this task is very difficult and requires a very sensitive network.

For this part, some modifications were made to the dataset. First, for each exercise (already in skeleton format) of all participants, all samples were divided into three different classes: OK, NOT OK and UNKNOWN. The sequences were cut to length or padded by skeletons initialised to zero. It should be mentioned that movement interpolation was tried but padding yielded better results in a range of one or two percentage points (mean class accuracy), then the sequences of each set (training, validation and test) were concatenated into a list and fed to the network So, with the modified dataset, a series of experiments were conducted, measuring the mean class accuracy and, most importantly, analysing the output of the loss function.

After analysing the training curves shown in Figure 5.3, some conclusions can be drawn. In (a), the loss curves of the validation and training sets show similar behaviour and have a good rate of convergence until they reach a plateau near the 200th epoch. A look at the accuracy curve shows that the achieved accuracy is quite good, with a mean class accuracy of 96.45%. On the other hand, if the graphs b), c) and d), are analysed, the same conclusions can not be drawn. c) and d) are clearly under fitting, the results of the accuracy testing just shows that the model does not learn and that there is a very high class imbalance. In b), the model is slightly underfitting and the raggedness of the curve shows that the training dataset is not representative. This shows that the number of samples has a large information gap.

The inference process is quite fast, about two hundredths of a second, which allows for a very fast prediction, provided the skeleton landmarks are already extracted and the subsequent skeleton is preprocessed.

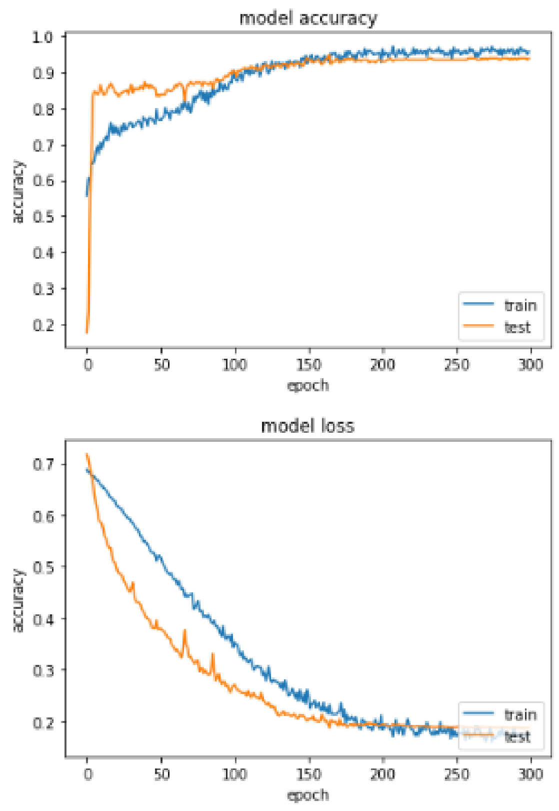
5.3.3 Motion Prediction and Feedback

For the motion prediction task, the MPJPE error function was modified a little. Instead of calculating the mean position error per joint, the position error the joint position error was calculated, making the error per joint accessible. The notation used bellow is the same as in the MPJPE, equation 4.16:

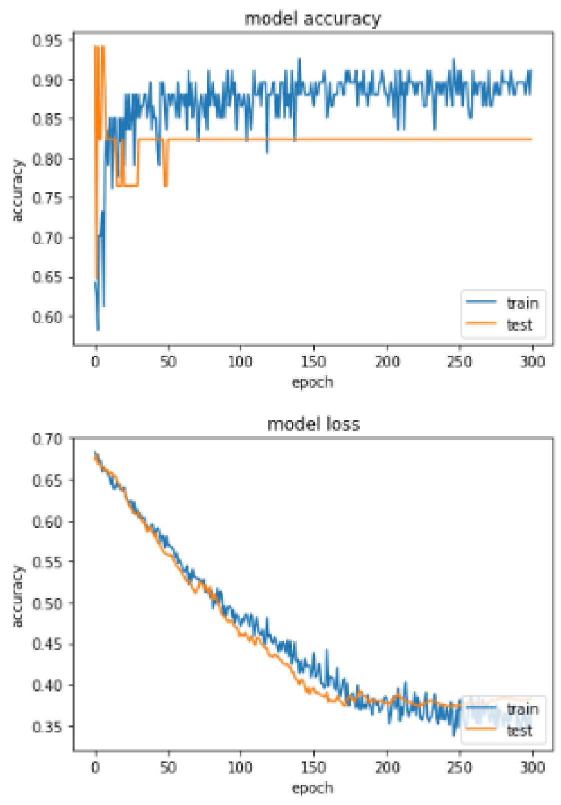
$$L_{JPE} = \frac{1}{V(T+K)} ||\hat{x}_{vk} - x_{vk}||_2 \tag{5.7}$$

Keeping the same notation of the equation 4.16, instead of calculating the mean distance of all the joints through all time-steps, the positional Euclidean error will be calculated for each joint, for each time-step. This way, according and with prior knowledge of where the rehabilitation work will focus limbwise, or the INPACT project, it is assumed that there is knowledge on what part of the body will be assessed and the exercise to be executed, i.e. the rehabilitation that includes walking/running for post-operative or even neurodegenerative diseases like multiple sclerosis rehab, the focus will be on the legs while other exercises like squats will include both the legs and the spine.

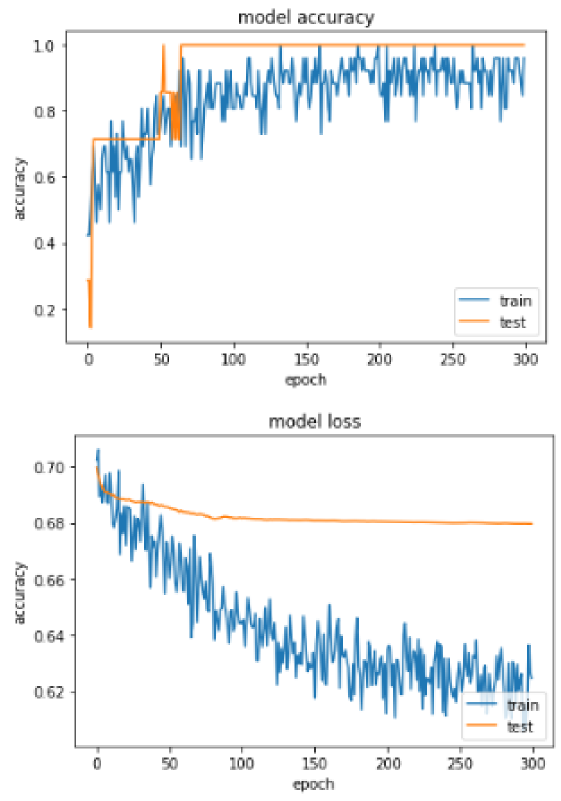
Figure 5.4 shows the sequence of 25 frames, corresponding to almost 1 second of prediction, focusing on the error of the legs. The model used was the STS-GCN by [25] using 3D Cartesian Coordinates with 10 frames of movement context. The error was thresholded for a better visual inspection of the prediction. In the image the striped black lines represent the ground truth, while the prediction is colour coded by several thresholds of accuracy, green meaning very good, yellow meaning good, orange meaning mediocre and red meaning the observed movement is not following the predicted average movement. These thresholds will enable the system to provide meaningful instructions to the user. This removes useless joints for the analysis and provides an easier visual feedback towards the user.



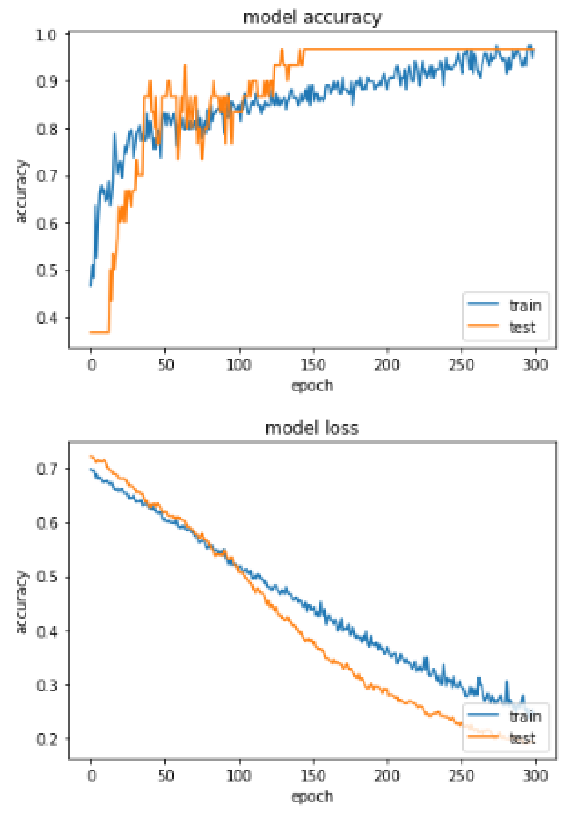
(a) Training and validation accuracy and losses wrt squats



(b) Training and validation accuracy and losses wrt sit-ups



(c) Training and validation accuracy and losses wrt push-ups



(d) Training and validation accuracy and losses wrt jumping-jacks

Figure 5.3: Results for an experiment in the conditions above stated in 4.2.1

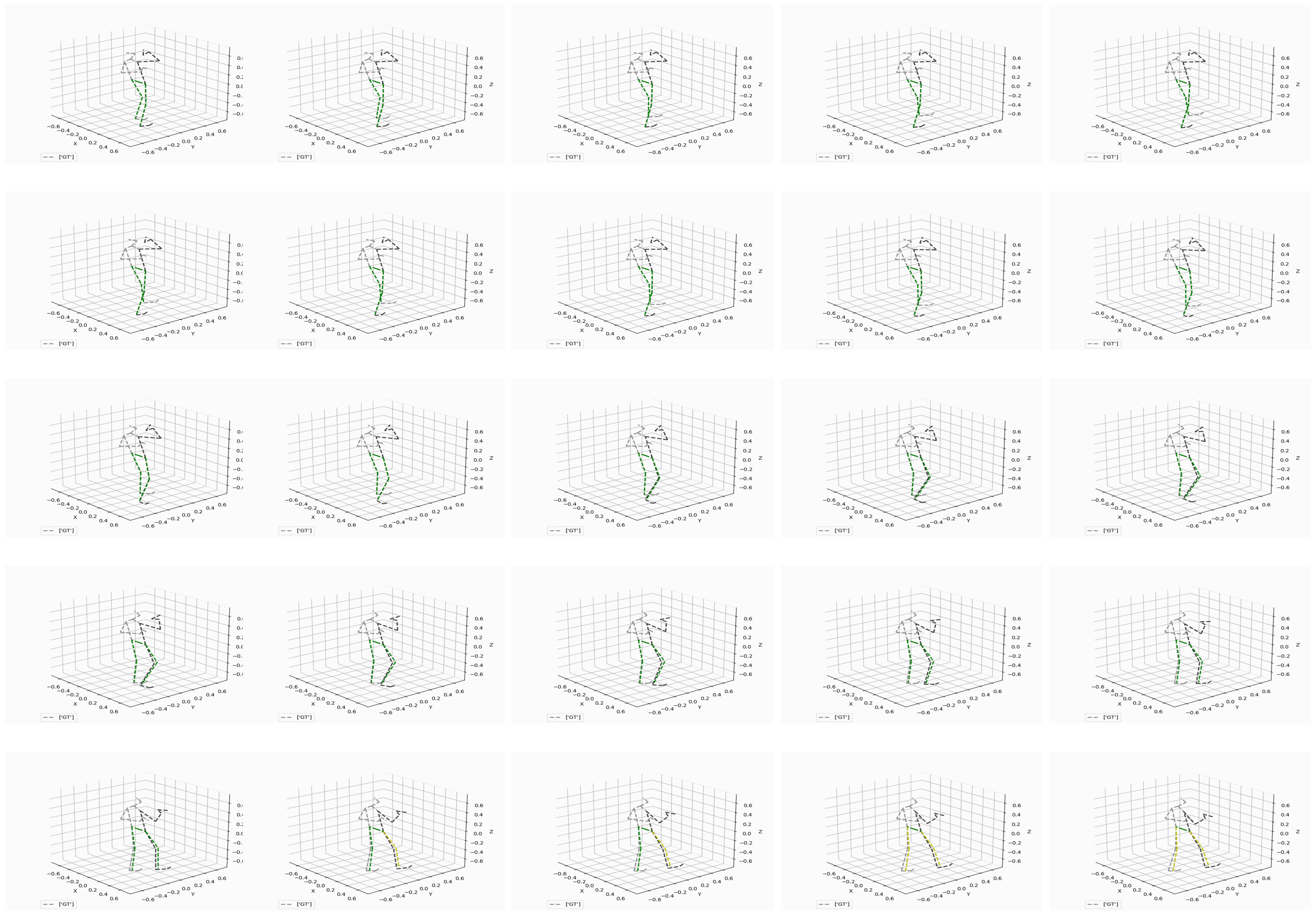


Figure 5.4: Sequence of 25 images in a walking exercise

One second of prediction time is quite unreasonable for real-world applications, but it shows the robustness of the architecture. The speed of the movement was also changed to see if it had any effect on the prediction. This was done by trying the same action but subsampling it from 50% to 25%, increasing the distance between joints and between frames. The model proved to be robust enough to make the necessary corrections to the distance.

By an empirical analysis of the aforementioned Figure 5.4 it can be seen that the model is fairly accurate, provided the motion is linear and the context corresponds to the motion to be executed. As expected, the prediction increasingly deviates from the actual situation in the last few frames.

In order not to underestimate the importance of context, another experiment was conducted. At the beginning of this exercise, the performer stood still. Only after a few frames of prediction does the performer begin to move. This can be demonstrated empirically by analysing Figure 5.5. Another fact regarding context is that the more context samples there are, the better the prediction. Thus, more complex features such as a better approximation of speed can be extracted.

It should also be noted that the joint position error increases as the number of prediction samples increases. This happens because the error accumulates after each time step and the observed ground truth deviates more and more from the prediction. Therefore, it is best not to overstretch the capabilities of the model by increasing the number of predicted samples. Another important point is that the models are not able to produce results in real time. As for the language used, Python is not known for being the fastest language. C, C++ and Rust are the fastest, yet Python provides an amazing backbone with many machine learning oriented functions and libraries. Still, the entire inference process takes between two and five seconds of processing time to generate the predicted sequence, its associated error and saving the drawn images. Provided the landmarks are already extracted and the sequence are preprocessed.

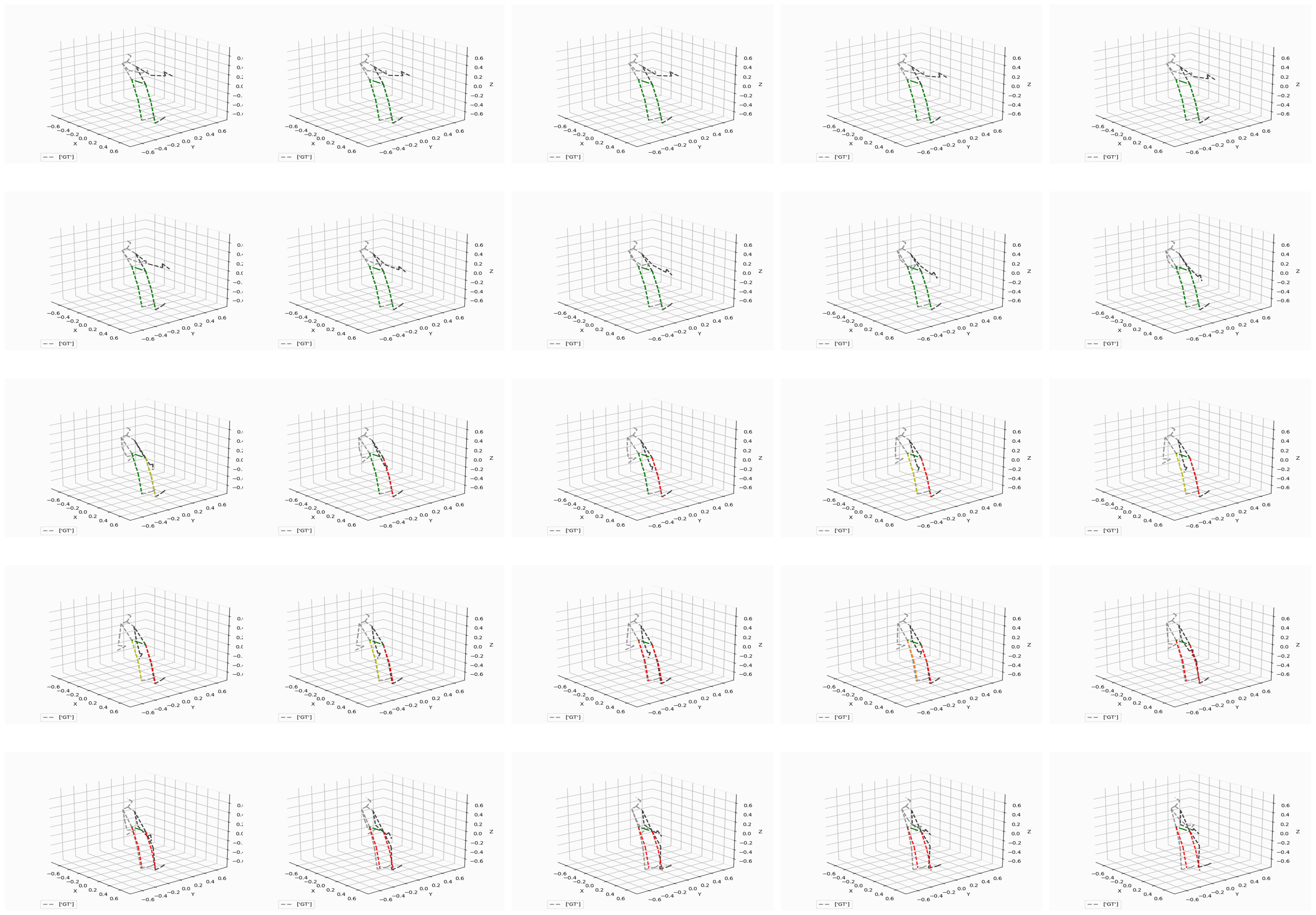


Figure 5.5: Sequence of 25 walking frames proving the importance of historic context

6 Conclusion and Future Work

With the ever-increasing need for computer assisted aids in almost all medicine fields and the massive bump in the public interest for it in this post Covid-19 world, the search for novel ways to assist humans in saving lives and improving the quality of life have taken a front row seat in the development priority. As stated before in the context of the INPACT project, the goal was to implement a full-stack project, to help in the execution of rehabilitation exercises, both in-house and out-house. The novelty of the project arises due to the fact that this has not been done for rehab purposes and does not include a full-fledged working project.

In this thesis, a catalogue of back-end methods to help with physical rehabilitation has been presented and implemented. These can be divided into two categories: Action Recognition, where correctly and incorrectly performed exercises were tested, and Movement Prediction, where two different machine learning models create a fixed-size sequence (the predicted sequence) given a time sequence as context. The observed sequence (the ground truth) and the predicted sequence were compared, and the total error measured joint by joint.

On the proposed pipeline. A sequence of RGB images is used to extract skeletal landmarks, which are normalised and subsampled to an appropriate length. Then the context sequence is collected and fed to the action recognition model or the motion prediction model. Then, if the former, is chosen the data is sifted through a Bidirectional LSTM with a self-attention mechanism, and depending on the goodness of execution, a classification is made. If the latter, is chosen, one of two options: Extract the landmarks as 3D Cartesian coordinates or extract them as 3D Euler angles and further process them to exponential map format. Although this guarantees better results due to the lack of gimbal locking (loss of one degree of freedom), the raw extraction of 3D coordinates results in faster computation time as there is no need for further pre-processing with negligible degradation of the extracted metrics.

Second, in motion prediction, the joints of the skeleton are fed by one of two different types of Convolutional Graph Neural Networks, one a space-time separable graph convolutional network and an anchor-based graph convolutional network. These can extract latent features within the graph representation of the input data and use them to obtain a predicted sequence by iterating them in each step until the last predicted frame.

Third, these sequences are used to determine an error metric by comparing it to the observed sequence (the ground truth) and calculating an error variable. A skeleton is created and given to the user as feedback.

Moreover, the motion prediction architectures have some very interesting properties. First, they are quite robust to changes in speed and sampling rate. The networks adjust the output sequence according to the execution speed and sampling rate without the need for a dynamic time warping algorithm. Secondly, they are very sensitive to the given context. It is worth noting that the predicted sequence is much more accurate for longer context sequences than for shorter sequences.

These models were implemented in Python using the TensorFlow and PyTorch libraries. The results were measured in FPR and mean class accuracy in the case of the action detection model and in Mean Per Joint Position Error (MPJPE) loss function variations in the case of motion prediction. All these methods were compared with other state-of-the-art methods.

6.1 Future Work

Improvements to the current system need to be made to further validate the methods, and some other improvements to the models need to be made. Firstly, more datasets with more diverse and complicated movements need to be tested, specially those with workout or even rehabilitation exercises in mind.

Not only, but also, some improvements need to be made to the data preprocessing. For once, a inverse kinematics function that transforms 3D Cartesian Coordinates into 3D Euler and then to exponential map needs to be created, so a higher threshold of prediction accuracy is achieved.

There are still some other ways in which the overall system can be improved, either by implementing new features or by joining it with other systems

- Implementation of multi-person prediction and exercise analysis
- Addition of prediction of movement an analysis of more specific and intricate parts of

the body, like the spinal column

- Movement segmentation for repetition counter
- Add an overall scoring function for the action recognition model
- Due to current performance benchmarks, add a faster, less sophisticated model to make corrections without predicting the movement.

With these proposals and a broader array of datasets, these methods are surely to produce even better results and a more intuitive feedback. Not only this but the proposed system has the capability to be modified for other types of data and scenarios like in the automotive industry or meteorology.

Bibliography

[1] C. Ionescu, D. Papava, V. Olaru, and C. Sminchisescu, “Human3.6m: Large scale datasets and predictive methods for 3d human sensing in natural environments”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 7, pp. 1325–1339, Jul. 2014.

[2] N. Mahmood, N. Ghorbani, N. F. Troje, G. Pons-Moll, and M. J. Black, *Amass: Archive of motion capture as surface shapes*, 2019. arXiv: `1904.03278 [cs.CV]`.

[3] J. Bocas, “The digitalization of medicine”, in *Digital Health and Wearables*, 2022.

[4] L. Rigamonti, C. L. Urs-Vito Albrecht, M. Tempel, B. Wolfarth, and D. A. Back, “Working group digitalisation. potentials of digitalization in sports medicine”, in *Current Sports Medicine Reports*, vol. 19, 2020, pp. 157–163. DOI: `DOI:10.1249/JSR.0000000000000704`.

[5] J. Manyika, S. Ramaswamy, S. Khanna, *et al.*, “Digital america: A tale of the haves and have-mores”, in *McKinsey Global Institute*, 2015. [Online]. Available: `https://www.mckinsey.com/industries/high-tech/our-insights`.

[6] M. Müschenich and L. Wamprecht, “Health 4.0 — how are we doing tomorrow?”, in *Bundesgesundheitsblatt Gesundheitsforschung Gesundheitsschutz*, vol. 9, 2018, ch. 585-606.

[7] B.Mesko, Z. Drobni, E. Benyei, *et al.*, “Digital health is a cultural transformation of traditional healthcare”, in *Mhealth*, 2017. DOI: `10.21037/mhealth.2017.08.07.`. [Online]. Available: `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5682364`.

[8] W. Frontera, L. Micheli, S. Herring, and J. Silver, “Medical management and rehabilitation”, in *Elsevier Inc*, 2007.

[9] H. Saner and v. d. V. E, “Ehealth in cardiovascular medicine: A clinical update”, in *Eurpean Journal of Preventive Cardiology*, vol. 23, 2016, ch. 5-12.

[10] D. Lupton, “The digitally engaged patient: Self-monitoring and self-care in the digital health era.”, in *Social Science and Medicine*, vol. 86, 2013.

[11] E. Sillence, P. Briggs, P. Harris, and L. Fishwick, “How do patients evaluate and make use of online health information?”, in *Social Science and Medicine*, vol. 64, 2007.

[12] Y. Liao, A. Vakanski, and M. Xian, “A deep learning framework for assessing physical rehabilitation exercises”, in *IEEE*, 2019, pp. 1–9.

[13] S. R. Machlin, J. Chevan, W. W. Yu, and M. W. Zodet, “Determinants of utilization and expenditures for episodes of ambulatory physical therapy among adults”, in *Physical Therapy*, vol. 91, 2011, pp. 1018–1029.

[14] R. Komatireddy, A. Chokshi, J. Basnett, M. Casale, D. Goble, and T. Shubert, “Quality and quantity of rehabilitation exercises delivered by a 3-d motion controlled camera: A pilot study”, in *International Journal of Physical Medicine and Rehabilitation*, vol. 2, 2014.

[15] “Intelligent platform for autonomous collaborative telerehabilitation”. [Online]. Available: `https://isr.uc.pt/index.php/projects/current-projects?task=showprojects.show$%5C%$28$%5C%$29&idProject=274`.

[16] M. Varandas and P. Peixoto, *3d pose and human shape estimation for autonomous telerehabilitation systems*, Master’s thesis, ISR, Departamento de Engenharia Eletrotécnica e Computadores, Universidade de Coimbra, Jul. 2023.

[17] J. Li, C. Xu, Z. Chen, S. Bian, L. Yang, and C. Lu, *Hybrik: A hybrid analytical-neural inverse kinematics solution for 3d human pose and shape estimation*, 2022. arXiv: `2011.14672 [cs.CV]`.

[18] O. Avci, O. Abdeljaber, S. Kiranyaz, M. Hussein, M. Gabbouj, and D. J. Inman, “A review of vibration-based damage detection in civil structures: From traditional methods to machine learning and deep learning applications”, *Mechanical Systems and Signal Processing*, vol. 147, p. 107 077, 2021, ISSN: 0888-3270. DOI: `https://doi.org/10.1016/j.ymssp.2020.107077`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0888327020304635`.

[19] K. O’Shea and R. Nash, “An introduction to convolutional neural networks”, in *School of Computing and Communications, Lancaster University, Lancashire*, 2015.

[20] Z. Liu, S. Wu, S. Jin, *et al.*, "Towards natural and accurate future motion prediction of humans and animals", in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 9996–10 004. DOI: 10.1109/CVPR.2019.01024.

[21] F. han, B. Reily, *et al.*, "Space-time representation of people based on 3d skeletal data: A review", in *CVIU*, vol. 158, 2017, pp. 85–105.

[22] D. Hoeim and S.Savarese, "Representations and techniques for 3d object recognition and scene interpretation", in *Synthesis Lectures on Artificial Intelligence and Machine Learning.Morgan & Claypool Publishers*, 2011.

[23] J. Martinez, M. J. Black, and J. Romero, "On human motion prediction using recurrent neural networks", 2017. arXiv: 1705.02445 [cs.CV].

[24] W. Mao, M. Liu, and M. Salzmann, "History repeats itself: Human motion prediction via motion attention", 2020. arXiv: 2007.11755 [cs.CV].

[25] T. Sofianos, A. Sampieri, L. Franco, and F. Galasso, "Space-time-separable graph convolutional network for pose forecasting", 2021. arXiv: 2110.04573 [cs.CV].

[26] A. Jain, A. R. Zamir, S. Savarese, and A. Saxena, "Structural-rnn: Deep learning on spatio-temporal graphs", 2016. arXiv: 1511.05298 [cs.CV].

[27] A. Vihya, "A comprehensive guide on human pose estimation". [Online]. Available: https://www.analyticsvidhya.com/blog/2022/01/a-comprehensive-guide-on-human-pose-estimation/.

[28] T. Vidvan, "Human pose estimation using opencv & python". [Online]. Available: https://techvidvan.com/tutorials/human-pose-estimation-opencv/.

[29] X. Yun and E. R. Bachmann, "Design, implementation, and experimental results of a quaternion-based kalman filter for human body motion tracking", *IEEE Transactions on Robotics*, vol. 22, no. 6, pp. 1216–1227, 2006. DOI: 10.1109/TRO.2006.886270.

[30] G. Panahandeh, N. Mohammadiha, A. Leijon, and P. Händel, "Continuous hidden markov model for pedestrian activity classification and gait analysis", *IEEE Transactions on Instrumentation and Measurement*, vol. 62, no. 5, pp. 1073–1083, 2013. DOI: 10.1109/TIM.2012.2236792.

[31] Y. Huang, K. Englehart, B. Hudgins, and A. Chan, "A gaussian mixture model based classification scheme for myoelectric control of powered upper limb prostheses", *IEEE Transactions on Biomedical Engineering*, vol. 52, no. 11, pp. 1801–1811, 2005. DOI: 10.1109/TBME.2005.856295.

[32] A. Vakanski, I. Mantegh, A. Irish, and F. Janabi-Sharifi, "Trajectory learning for robot programming by demonstration using hidden markov model and dynamic time warping", *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 42, no. 4, pp. 1039–1052, 2012. DOI: 10.1109/TSMCB.2012.2185694.

[33] D. Biswas, Z. Ye, E. B. Mazomenos, M. Jöbges, and K. Maharatna, "Cordic framework for quaternion-based joint angle computation to classify arm movements", in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2018, pp. 1–5. DOI: 10.1109/ISCAS.2018.8350967.

[34] E. Mazomenos, D. Biswas, A. Cranny, *et al.*, "Detecting elementary arm movements by tracking upper limb joint angles with marg sensors", *IEEE journal of biomedical and health informatics*, vol. 20, May 2015. DOI: 10.1109/JBHI.2015.2431472.

[35] M. del Pra, "An overview of the architecture and the implementation details of the most important deep learning algorithms for time series forecasting", in *Time Series Forecasting with Deep Learning and Attention Mechanism.* [Online]. Available: https://towardsdatascience.com/time-series-forecasting-with-deep-learning-and-attention-mechanism-2d001fc871fc.

[36] A. Ohri, S. Agrawal, and G. Chaudhary, "On-device realtime pose estimation & correction", in *International Journal of Advances in Engineering and Management (IJAEM)*, vol. 3, 2021.

[37] J. Zhang, "Dynamic time warping:explanation and code implementation.". [Online]. Available: https://towardsdatascience.com/dynamictime-%20warping-3933f25fcdd.

[38] S. Bhan, "What is dynamic time warping?". [Online]. Available: https://medium.com/mlearning-ai/what-is-dynamic-time-warping-253a6880ad12.

[39] H. Salehinejad, S. Sankar, J. Barfett, E. Colak, and S. Valaee, *Recent advances in recurrent neural networks*, 2018. arXiv: 1801.01078 [cs.NE].

[40] LeCun, Bengio, and Hinton, "Deep learning", in *Nature*, vol. 521, 2015, pp. 436–444.

[41] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult", 2, vol. 5, 1994, pp. 157–166. DOI: 10.1109/72.279181.

[42] T. Mikolov, A. Joulin, S. Chopra, M. Mathieu, and M. Ranzato, *Learning longer memory in recurrent neural networks*, 2015. arXiv: 1412.7753 [cs.NE].

[43] K. Brind, "What is machine learning? definition, types, and examples.", in *Certificate in Quantitative Finance Institute*, 2022.

[44] K. D. Foote, *A brief history of machine learning*, May 2023. [Online]. Available: https://www.dataversity.net/a-brief-history-of-machine-learning/#.

[45] Z.-H. Zhou, *Machine Learning.* Nanjing: Springer Signature, 2021, ISBN: 9811519676, 9789811519673.

[46] D. O. Hebb, *The organization of behavior: A neuropsychological theory.* New York: Wiley, Jun. 1949, ISBN: 0-8058-4300-0.

[47] S. Grossberg, "Recurrent neural networks", in *Scholarpedia*, vol. 8, 2013. DOI: 10.4249/scholarpedia.1888.

[48] D. Chortarias, *Human activity recognition with deep learning*, 2021.

[49] L. C. Jain and L. R. Medsker, *Recurrent Neural Networks: Design and Applications*, 1st. USA: CRC Press, Inc., 1999, ISBN: 0849371813.

[50] L. Fausett, *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications* (Prentice-Hall international editions). Prentice-Hall, 1994, ISBN: 9780133341867. [Online]. Available: https://books.google.pt/books?id=ONylQgAACAAJ.

[51] S. Haykin, *Neural Networks: A Comprehensive Foundation.* Prentice Hall, 1999.

[52] aishwarya.27, "Introduction to recurrent neural network". [Online]. Available: https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/.

[53] S. Amidi, "Recurrent neural networks cheatsheet". [Online]. Available: https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks.

[54] J. Nabi, "Recurrent neural networks (rnns)", in *Implementing an RNN from scratch in Python.*. [Online]. Available: https://towardsdatascience.com/recurrent-neural-networks-rnns-3f06d7653a85.

[55] J. Delua, "Supervised vs. unsupervised learning: What's the difference?". [Online]. Available: https://www.ibm.com/cloud/blog/supervised-vs-unsupervised-learning.

[56] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks", *Neural Information Processing Systems*, vol. 25, Jan. 2012. DOI: 10.1145/3065386.

[57] R. Kwiatkowski, "Gradient descent algorithm — a deep dive", in *The Gradient Descent method lays the foundation for machine learning and deep learning techniques. Let's explore how does it work, when to use it and how does it behave for various functions.*, 2021. [Online]. Available: https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21.

[58] J. Watt, R. Borhani, and A. K. Katsaggelos, *Machine Learning Refined: Foundations, Algorithms, and Applications.* Cambridge University Press, 2016, ISBN: 9781316402276. DOI: 10.1017/CBO9781316402276. [Online]. Available: https://doi.org/10.1017/CBO9781316402276.

[59] G. Mayanglambam, "Deep learning optimizers", in *SGD with momentum, Adagrad, Adadelta, Adam optimizer*, 2020. [Online]. Available: https://towardsdatascience.com/deep-learning-optimizers-436171c9e23f.

[60] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016, http://www.deeplearningbook.org.

[61] S. Hochreiter and J. Schmidhuber, "Long short-term memory", *Neural computation*, vol. 9, pp. 1735–80, Dec. 1997. DOI: 10.1162/neco.1997.9.8.1735.

[62] C. Olah, "Understanding lstm networks", 2015. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[63] F. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm", in *1999 Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470)*, vol. 2, 1999, 850–855 vol.2. DOI: 10.1049/cp:19991218.

[64] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, "Dive into deep learning", *CoRR*, vol. abs/2106.11342, 2021. arXiv: 2106.11342. [Online]. Available: https://arxiv.org/abs/2106.11342.

[65] Y. Tamura, "Lstm back propagation: Following the flows of variables", in *Artificial Intelligence, Data Science, Deep Learning, Main Category*, 2020. [Online]. Available: https://data-science-blog.com/blog/2020/09/07/back-propagation-of-lstm/.

[66] M. Alhamid, “Lstm and bidirectional lstm for regression”, in *Learn how to use Long Short-Term Memory Networks for regression problems*, 2021. [Online]. Available: https://towardsdatascience.com/lstm-and-bidirectional-lstm-for-regression-4fddf910c655.

[67] E. Zvornicanin, “Differences between bidirectional and unidirectional lstm”. [Online]. Available: https://www.baeldung.com/cs/bidirectional-vs-unidirectional-lstm.

[68] N. Malingan, “Attention mechanism in deep learning”. [Online]. Available: https://www.scaler.com/topics/deep-learning/attention-mechanism-deep-learning/.

[69] Z. Niu, G. Zhong, and H. Yu, “A review on the attention mechanism of deep learning”, *Neurocomputing*, vol. 452, pp. 48–62, 2021, ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2021.03.091. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S092523122100477X.

[70] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, *Attention is all you need*, 2017. arXiv: 1706.03762 [cs.CL].

[71] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, *A convolutional neural network for modelling sentences*, 2014. arXiv: 1404.2188 [cs.CL].

[72] B. Warner, “Tinkering with attention pooling”, in *Improving Upon Learned Aggregation.* [Online]. Available: https://benjaminwarner.dev/2022/07/14/tinkering-with-attention-pooling.

[73] E. A. Nadaraya, “On estimating regression”, *Theory of Probability & Its Applications*, vol. 9, no. 1, pp. 141–142, 1964. DOI: 10.1137/1109020. [Online]. Available: https://doi.org/10.1137/1109020.

[74] G. S. Watson, “Smooth regression analysis”, *Sankhyā: The Indian Journal of Statistics, Series A (1961-2002)*, vol. 26, no. 4, pp. 359–372, 1964, ISSN: 0581572X. [Online]. Available: http://www.jstor.org/stable/25049340 (visited on 05/29/2023).

[75] J. F. Kolen and S. C. Kremer, “Gradient flow in recurrent nets: The difficulty of learning longterm dependencies”, in *A Field Guide to Dynamical Recurrent Networks.* 2001, pp. 237–243. DOI: 10.1109/9780470544037.ch14.

[76] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, “A comprehensive survey on graph neural networks”, *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, 2021. DOI: 10.1109/TNNLS.2020.2978386.

[77] J. Zhou, G. Cui, S. Hu, *et al.*, *Graph neural networks: A review of methods and applications*, 2021. arXiv: 1812.08434 [cs.LG].

[78] “Https://web.stanford.edu/class/cs224w/slides/08-gnn.pdf”.

[79] M. M. Bronstein, J. Bruna, T. Cohen, and P. Veličković, *Geometric deep learning: Grids, groups, graphs, geodesics, and gauges*, 2021. arXiv: 2104.13478 [cs.LG].

[80] Y. Shastri, “A beginner’s guide to graph neural networks”, in *What are Graph Neural Networks (GNN)? Learn more about their architecture, applications in computer vision, and the reasons for their increasing popularity.*, 2022. [Online]. Available: https://www.v7labs.com/blog/graph-neural-networks-guide.

[81] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, “The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains”, *IEEE Signal Processing Magazine*, vol. 30, no. 3, pp. 83–98, 2013. DOI: 10.1109/MSP.2012.2235192.

[82] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, *Spectral networks and locally connected networks on graphs*, 2014. arXiv: 1312.6203 [cs.LG].

[83] D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, *et al.*, *Convolutional networks on graphs for learning molecular fingerprints*, 2015. arXiv: 1509.09292 [cs.LG].

[84] A. Micheli, “Neural network for graphs: A contextual constructive approach”, *IEEE Transactions on Neural Networks*, vol. 20, no. 3, pp. 498–511, 2009. DOI: 10.1109/TNN.2008.2010350.

[85] D. Bahdanau, K. Cho, and Y. Bengio, *Neural machine translation by jointly learning to align and translate*, 2016. arXiv: 1409.0473 [cs.CL].

[86] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, *Convolutional sequence to sequence learning*, 2017. arXiv: 1705.03122 [cs.CL].

[87] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, *Graph attention networks*, 2018. arXiv: 1710.10903 [stat.ML].

[88] H. Coskun, D. J. Tan, S. Conjeti, N. Navab, and F. Tombari, *Human motion analysis with deep metric learning*, 2018. arXiv: 1807.11176 [cs.CV].

[89] A. Newell, K. Yang, and J. Deng, *Stacked hourglass networks for human pose estimation*, 2016. arXiv: `1603.06937 [cs.CV]`.

[90] C. Zhao, J. G. Han, and X. Xu, "Cnn and rnn based neural networks for action recognition", *Journal of Physics: Conference Series*, vol. 1087, no. 6, p. 062 013, Sep. 2018. DOI: `10.1088/1742-6596/1087/6/062013`. [Online]. Available: `https://dx.doi.org/10.1088/1742-6596/1087/6/062013`.

[91] S. W. Pienaar and R. Malekian, *Human activity recognition using lstm-rnn deep neural network architecture*, 2019. arXiv: `1905.00599 [cs.LG]`.

[92] T. K. Vintsyuk, "Speech discrimination by dynamic programming", *Cybernetics*, vol. 4, no. 1, pp. 52–57, 1968, Russian Kibernetika 4(1):81-88 (1968).

[93] E. J. Keogh and M. J. Pazzani, "Derivative dynamic time warping", in *Proceedings of the 2001 SIAM International Conference on Data Mining (SDM)*, pp. 1–11. DOI: `10.1137/1.9781611972719.1`. eprint: `https://epubs.siam.org/doi/pdf/10.1137/1.9781611972719.1`. [Online]. Available: `https://epubs.siam.org/doi/abs/10.1137/1.9781611972719.1`.

[94] D. J. Berndt and J. Clifford, "Using dynamic time warping to find patterns in time series", in *KDD Workshop*, 1994.

[95] C. A. Ratanamahatana and E. Keogh, "Making time-series classification more accurate using learned constraints", in *Proceedings of the 2004 SIAM International Conference on Data Mining (SDM)*, pp. 11–22. DOI: `10.1137/1.9781611972740.2`. eprint: `https://epubs.siam.org/doi/pdf/10.1137/1.9781611972740.2`. [Online]. Available: `https://epubs.siam.org/doi/abs/10.1137/1.9781611972740.2`.

[96] Y. Zheng, Q. Liu, E. Chen, J. L. Zhao, L. He, and G. Lv, "Convolutional nonlinear neighbourhood components analysis for time series classification", in *Advances in Knowledge Discovery and Data Mining*, T. Cao, E.-P. Lim, Z.-H. Zhou, T.-B. Ho, D. Cheung, and H. Motoda, Eds., Cham: Springer International Publishing, 2015, pp. 534–546, ISBN: 978-3-319-18032-8.

[97] W. Pei, D. M. J. Tax, and L. van der Maaten, *Modeling time series similarity with siamese recurrent networks*, 2016. arXiv: `1603.04713 [cs.CV]`.

[98] A. López-Méndez, J. Gall, J. Casas, and L. Van Gool, "Metric learning from poses for temporal clustering of human motion", Sep. 2012. DOI: `10.5244/C.26.49`.

[99] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality reduction by learning an invariant mapping", in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 2, 2006, pp. 1735–1742. DOI: `10.1109/CVPR.2006.100`.

[100] F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A unified embedding for face recognition and clustering", in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, Jun. 2015. DOI: `10.1109/cvpr.2015.7298682`. [Online]. Available: `https://doi.org/10.1109%2Fcvpr.2015.7298682`.

[101] M. Schultz and T. Joachims, "Learning a distance metric from relative comparisons", in *Advances in Neural Information Processing Systems*, S. Thrun, L. Saul, and B. Schölkopf, Eds., vol. 16, MIT Press, 2003. [Online]. Available: `https://proceedings.neurips.cc/paper_files/paper/2003/file/d3b1fb02964aa64e257f9f26a31f72cf-Paper.pdf`.

[102] S. Xu, Y.-X. Wang, and L.-Y. Gui, "Diverse human motion prediction guided by multi-level spatial-temporal anchors", in *Lecture Notes in Computer Science*, Springer Nature Switzerland, 2022, pp. 251–269. DOI: `10.1007/978-3-031-20047-2_15`. [Online]. Available: `https://doi.org/10.1007%2F978-3-031-20047-2_15`.

[103] X. Zhang, F. X. Yu, S. Kumar, and S.-F. Chang, "Learning spread-out local feature descriptors", in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 4605–4613. DOI: `10.1109/ICCV.2017.492`.

[104] J. Goldberger, G. E. Hinton, S. Roweis, and R. R. Salakhutdinov, "Neighbourhood components analysis", in *Advances in Neural Information Processing Systems*, L. Saul, Y. Weiss, and L. Bottou, Eds., vol. 17, MIT Press, 2004. [Online]. Available: `https://proceedings.neurips.cc/paper_files/paper/2004/file/42fe880812925e520249e808937738d2-Paper.pdf`.

[105] K. Fragkiadaki, S. Levine, P. Felsen, and J. Malik, *Recurrent network models for human dynamics*, 2015. arXiv: `1508.00271 [cs.CV]`.

[106] W. Mao, M. Liu, M. Salzmann, and H. Li, *Learning trajectory dependencies for human motion prediction*, 2020. arXiv: `1908.05436 [cs.CV]`.

[107] J. Bütepage, M. Black, D. Kragic, and H. Kjellström, *Deep representation learning for human motion prediction and classification*, 2017. arXiv: `1702.07486 [cs.CV]`.

[108] I. Sutskever, O. Vinyals, and Q. V. Le, *Sequence to sequence learning with neural networks*, 2014. arXiv: 1409.3215 [cs.CL].

[109] S. Yan, Y. Xiong, and D. Lin, *Spatial temporal graph convolutional networks for skeleton-based action recognition*, 2018. arXiv: 1801.07455 [cs.CV].

[110] F. Chollet, *Xception: Deep learning with depthwise separable convolutions*, 2017. arXiv: 1610.02357 [cs.CV].

[111] C. Szegedy, W. Liu, Y. Jia, *et al.*, *Going deeper with convolutions*, 2014. arXiv: 1409.4842 [cs.CV].

[112] W. Hu, M. Fey, H. Ren, M. Nakata, Y. Dong, and J. Leskovec, *Ogb-lsc: A large-scale challenge for machine learning on graphs*, 2021. arXiv: 2103.09430 [cs.LG].

[113] W. Hu, M. Fey, M. Zitnik, *et al.*, *Open graph benchmark: Datasets for machine learning on graphs*, 2021. arXiv: 2005.00687 [cs.LG].

[114] V. P. Dwivedi, C. K. Joshi, A. T. Luu, T. Laurent, Y. Bengio, and X. Bresson, *Benchmarking graph neural networks*, 2022. arXiv: 2003.00982 [cs.LG].

[115] C. Ying, T. Cai, S. Luo, *et al.*, *Do transformers really perform bad for graph representation?*, 2021. arXiv: 2106.05234 [cs.LG].

[116] L. Zhao, Y. Song, C. Zhang, *et al.*, "T-GCN: A temporal graph convolutional network for traffic prediction", *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 9, pp. 3848–3858, Sep. 2020. DOI: 10.1109/tits.2019.2935152. [Online]. Available: https://doi.org/10.1109%2Ftits.2019.2935152.

[117] S. Yun, M. Jeong, R. Kim, J. Kang, and H. J. Kim, *Graph transformer networks*, 2020. arXiv: 1911.06455 [cs.LG].

[118] H. Qiang, Z. Guo, S. Xie, and X. Peng, *Mstformer: Motion inspired spatial-temporal transformer with dynamic-aware attention for long-term vessel trajectory prediction*, 2023. arXiv: 2303.11540 [cs.LG].

[119] S. Hu, L. Shen, Y. Zhang, Y. Chen, and D. Tao, *On transforming reinforcement learning by transformer: The development trajectory*, 2023. arXiv: 2212.14164 [cs.LG].

[120] R. Vemulapalli, F. Arrate, and R. Chellappa, "Human action recognition by representing 3d skeletons as points in a lie group", in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 588–595. DOI: 10.1109/CVPR.2014.82.

[121] F. S. Grassia, "Practical parameterization of rotations using the exponential map", *Journal of Graphics Tools*, vol. 3, no. 3, pp. 29–48, 1998. DOI: 10.1080/10867651.1998.10487493. eprint: https://doi.org/10.1080/10867651.1998.10487493. [Online]. Available: https://doi.org/10.1080/10867651.1998.10487493.

[122] J. Solà, J. Deray, and D. Atchuthan, *A micro lie theory for state estimation in robotics*, 2021. arXiv: 1812.01537 [cs.RO].

[123] C. Xu, L. N. Govindarajan, Y. Zhang, and L. Cheng, *Lie-x: Depth image based articulated object pose estimation, tracking, and action recognition on lie groups*, 2016. arXiv: 1609.03773 [cs.CV].

[124] C. Bregler and J. Malik, "Tracking people with twists and exponential maps", in *Proceedings. 1998 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No.98CB36231)*, 1998, pp. 8–15. DOI: 10.1109/CVPR.1998.698581.

[125] J. Batista, "Prozis challenge", 2019. [Online]. Available: https://www.isr.uc.pt/index.php/projects/past-projects?task=showprojects.show%28%29&idProject=219.

[126] B. Ferreira, P. Menezes, and J. Batista, "Transformers for workout video segmentation", in *2022 IEEE International Conference on Image Processing (ICIP)*, 2022, pp. 3470–3474. DOI: 10.1109/ICIP46576.2022.9897194.

[127] A. Mishchuk, D. Mishkin, F. Radenovic, and J. Matas, *Working hard to know your neighbor's margins: Local descriptor learning loss*, 2018. arXiv: 1705.10872 [cs.CV].

[128] Y. Movshovitz-Attias, A. Toshev, T. K. Leung, S. Ioffe, and S. Singh, *No fuss distance metric learning using proxies*, 2017. arXiv: 1703.07464 [cs.CV].

[129] F. Zhou and F. De la Torre, "Generalized canonical time warping", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 2, pp. 279–294, 2016. DOI: 10.1109/TPAMI.2015.2414429.

[130] C. Laurent, G. Pereyra, P. Brakel, Y. Zhang, and Y. Bengio, "Batch normalized recurrent neural networks", in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016, pp. 2657–2661. DOI: 10.1109/ICASSP.2016.7472159.

[131] Z. Lin, M. Feng, C. N. dos Santos, *et al.*, *A structured self-attentive sentence embedding*, 2017. arXiv: 1703.03130 [cs.CL].

[132] S. P. Singh, M. K. Sharma, A. Lay-Ekuakille, D. Gangwar, and S. Gupta, "Deep ConvLSTM with self-attention for human activity decoding using wearable sensors", *IEEE Sensors Journal*, vol. 21, no. 6, pp. 8575–8582, Mar. 2021. DOI: 10.1109/jsen.2020.3045135. [Online]. Available: https://doi.org/10.1109%2Fjsen.2020.3045135.

[133] Y. Yuan and K. Kitani, *Diverse trajectory forecasting with determinantal point processes*, 2019. arXiv: 1907.04967 [cs.CV].

[134] Y. Yuan and K. Kitani, *Dlow: Diversifying latent flows for diverse human motion prediction*, 2020. arXiv: 2003.08386 [cs.CV].

[135] W. Mao, M. Liu, and M. Salzmann, *Generating smooth pose sequences for diverse human motion prediction*, 2022. arXiv: 2108.08422 [cs.CV].

[136] C. mellon university - cmu graphics lab - motion capture library, *Http://mocap.cs.cmu.edu/*, 2010.

[137] D. J. Sutherland, H.-Y. Tung, H. Strathmann, *et al.*, *Generative models and model criticism via optimized maximum mean discrepancy*, 2021. arXiv: 1611.04488 [stat.ML].

[138] X. Li, H. Li, H. Joo, Y. Liu, and Y. Sheikh, *Structure from recurrent motion: From rigidity to recurrency*, 2018. arXiv: 1804.06510 [cs.CV].

[139] M. Hassan, D. Ceylan, R. Villegas, *et al.*, *Stochastic scene-aware motion prediction*, 2021. arXiv: 2108.08284 [cs.CV].

[140] Z. Liu, P. Su, S. Wu, *et al.*, "Motion prediction using trajectory cues", in *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 13 279–13 288. DOI: 10.1109/ICCV48922.2021.01305.

[141] M. Luber, J. A. Stork, G. D. Tipaldi, and K. O. Arras, "People tracking with human motion predictions from social forces", in *2010 IEEE International Conference on Robotics and Automation*, 2010, pp. 464–469. DOI: 10.1109/ROBOT.2010.5509779.

[142] M. Li, S. Chen, Y. Zhao, Y. Zhang, Y. Wang, and Q. Tian, *Dynamic multiscale graph neural networks for 3d skeleton-based human motion prediction*, 2020. arXiv: 2003.08802 [cs.CV].

[143] T. N. Kipf and M. Welling, *Semi-supervised classification with graph convolutional networks*, 2017. arXiv: 1609.02907 [cs.LG].

[144] W. Liu, D. Anguelov, D. Erhan, *et al.*, “SSD: Single shot MultiBox detector”, in *Computer Vision – ECCV 2016*, Springer International Publishing, 2016, pp. 21–37. DOI: `10.1007/978-3-319-46448-0_2`. [Online]. Available: `https://doi.org/10.1007%2F978-3-319-46448-0_2`.

[145] H. S. Koppula and A. Saxena, “Anticipating human activities for reactive robotic response”, in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 2071–2071. DOI: `10.1109/IROS.2013.6696634`.

[146] C. Li, Z. Zhang, W. S. Lee, and G. H. Lee, *Convolutional sequence to sequence model for human dynamics*, 2018. arXiv: `1805.00655 [cs.CV]`.

[147] B. C. Hall, *An elementary introduction to groups and representations*, 2000. arXiv: `math-ph/0005032 [math-ph]`.

[148] R. M. Murray, S. S. Sastry, and L. Zexiang, *A Mathematical Introduction to Robotic Manipulation*, 1st. USA: CRC Press, Inc., 1994, ISBN: 0849379814.

[149] E. Gallo, *The so(3) and se(3) lie algebras of rigid body rotations and motions and their application to discrete integration, gradient descent optimization, and state estimation*, 2023. arXiv: `2205.12572 [cs.RO]`.

[150] Y. Ma, S. Soatto, J. Kosecka, and S. S. Sastry, *An Invitation to 3-D Vision: From Images to Geometric Models*. SpringerVerlag, 2003, ISBN: 0387008934.

[151] J. L. Blanco-Claraco, *A tutorial on* $\mathbf{SE}(3)$ *transformation parameterizations and on-manifold optimization*, 2022. arXiv: `2103.15980 [cs.RO]`.

Appendix A

Lie Group Theory and Exponential Maps

In this appendix, the focus of will be on the Special Euclidean Group 3 (*SE(3)*), which is the most valuable for this thesis. For further reading on the basic definitions and concepts, Brian C. Hall [147] explains it in greater detail and for further details on rigid body kinematics the book "A Mathematical Introduction to Robotic Manipulation" [148] provides a great insight.

SE(3), is the set of 4×4 matrices made denoted by

$$P(R, \overrightarrow{d}) = \begin{bmatrix} R & \overrightarrow{d} \\ 0 & 1 \end{bmatrix} \tag{A.1}$$

where $\overrightarrow{d} \in \mathcal{R}^3$ is the translation vector and $R \in \mathcal{R}^{3\times3}$ represents the 3×3 rotation matrix. The members of the *SE(3)* group act on the points points $z \in \mathcal{R}^3$, by applying a rotation and then a translation [20], [120], [123], [148], [149]:

$$\begin{bmatrix} R & \overrightarrow{d} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} z \\ 1 \end{bmatrix} = \begin{bmatrix} Rz + \overrightarrow{d} \\ 1 \end{bmatrix} \tag{A.2}$$

The members of this set engage in standard matrix multiplication, and when viewed geometrically, they can be arranged in a seamless manner to create a curved manifold with six dimensions. This arrangement grants them the characteristics of a Lie group [147], [149]. Within this group, the 4 by 4 identity matrix I4 is included and recognized as the group's identity element.

The Lie algebra of *SE(3)*, denoted as $se(3)$, is the tangent plane to the identity element I4 of SE(3). It consists of 6 dimensions and is represented by 4 by 4 matrices of the form $\begin{bmatrix} :U & \overrightarrow{w} \\ 0 & 0 \end{bmatrix}$, where U is a 3×3 skew-matrix and the vector $\overrightarrow{w} \in \mathcal{R}^3$ is once again the translation [120], [123], Thus

$$B = \begin{bmatrix} U & \overrightarrow{w} \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & -u_3 & u_2 & w_1 \\ u_3 & 0 & -u_1 & w_2 \\ -u_2 & u_1 & 0 & w_3 \\ 0 & 0 & 0 & 0 \end{bmatrix} \in \ se(3) \tag{A.3}$$

with the vector representation:

$$\text{vec}(B) = [u_1, u_2, u_3, w_1, w_2, w_3] \tag{A.4}$$

And with the closed form solution [150]:

$$U = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \frac{\theta}{2\text{sin}\theta} \begin{bmatrix} R(3,2) - R(2,3) \\ R(1,3) - R(3,1) \\ R(2,1) - R(1,2) \end{bmatrix} \tag{A.5}$$

where $\theta = \text{arcos}(\frac{Tr(R)-1)}{2})$ and:

$$w = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \left[\frac{(I_3 - R)U_\times + U\,U^T}{||U||}\right]^{-1} \tag{A.6}$$

The exponential map $\exp_{SE(3)} : se(3) \ \rightarrow \ SE(3)$ and the logarithm map $\log_{SE(3)} : SE(3) \ \rightarrow \ se(3)$ [20], [120], [123], are given by:

$$\begin{aligned} \exp_{SE(3)}(B) &= \mathbf{e}^B \\ \log_{SE(3)}(P) &= \mathbf{log}(P) \end{aligned} \tag{A.7}$$

e and **log** denoting the matrix exponential and logarithm, **e** which maps elements from the algebra to the manifold and determines the local structure of the manifold and **log** which maps elements from the manifold to the algebra [20], [120], [123], [148], [151].