DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2021

One rig to rule them all

KTH Master Thesis

Nuo Chen

KTH ROYAL INSTITUTE OF TECHNOLOGY
COMPUTER SCIENCE

Abstract

Various research has shown the potential and robustness of deep learning-based approaches to synthesise novel motions of 3D characters in virtual environments, such as video games and films. The models are trained with the motion data that is bound to the respective character skeleton (rig). It inflicts a limitation on the scalability and the applicability of the models since they can only learn motions from one particular rig (domain) and produce motions in that domain only. Transfer learning techniques can be used to overcome this issue and allow the models to both learn and produce motions irrespective of the rig configuration.

In this thesis, a study of three transfer learning techniques for an objective-driven motion generation network (OMG) is presented. Three approaches for achieving rig-agnostic-encoding (RAE) are proposed and experimented: **Feature encoding** using autoencoders to encode the pose data of rigs with different configuration to the latent space of the same dimension; **Feature clustering** using an additional neural network model to find an abstract representation of the pose data in the same feature space; **Feature selection** using a fixed subset of the features to represent the pose data, and relying on OMG for upsampling to the full-resolution pose.

By utilising RAE, the pose data from different rigs can be encoded to similar embeddings, whether feature-wise or dimension-wise. OMG predicts the pose for the next frame given the current pose and the local motion phase. The prediction is conditioned on the Euclidean distance to the target positions and the angular distance to the target rotations. The three RAE approaches are experimented with to investigate the performance improvement of OMG on new domains by warm starting the model with the pre-trained parameters of the model on a previous domain.

<Insert results and conclusion>

Keywords

Transfer learning, data-driven motion synthesis, objective-driven motion generation, rig-agnostic encoding, deep learning-based clustering model, procedural animation

Abstrakt

Nyckelord

Procedurell animation, animation anpassning, mål-driven rörelse generation, deep learning, local rörelse fas

Acknowledgements

I would like to express my sincere gratitude for my supervisors Oskar Blom, Johan Tunkrans, Kristoffer Jonsson and many others at EA Digital Illusions CE AB (DICE), for their strong support and valuable advice throughout the course of this project. I also want to thank my supervisor and examiner Christopher Peters and Pawel Herman for their greatest assistance that improved my research and writing. At last we want to thank Taqui Syed, Viktor Vitek and Viktor Meyer for their insightful and constructive feedback.

Authors

Nuo Chen
nuoc@kth.se
Stockholm, Sweden
Computer Science with specialisation in Data Science
KTH Royal Institute of Technology

Examiner

Pawel Herman
Stockholm, Sweden
KTH Royal Institute of Technology

Supervisor

Christopher Peters
Stockholm, Sweden
KTH Royal Institute of Technology

Contents

List of Figures

Nomenclature

Acronyms

AE	Autoencoder
C-model	Clustering layer (RBF / VAE / DEC)
DEC	Deep embedding clustering network
Dec	Decoder
Enc	Encoder
FC-CAT-OMG	OMG with AE, C-model and MoGen, where Ψ is concatenated with z before feeding to MoGen
FC-IN-OMG	OMG with AE, C-model and MoGen, where Ψ is used as input to MoGen

FE-OMG	OMG with only AE and MoGen
LSTM	Long short-term memory network
MLP	Multi-layer perceptron network
MoE	Mixture-of-experts network
MoGen	Motion generation network (MoE, LSTM)
OMG	Objective-driven motion generation network
RBF	Radial-basis-function network
VAE	Variational autoencoder

Data related symbols

K	Number of key joints
M	Dataset size
n	Input dimension
X, Y	Input and output variables
x, y	Input and output vectors
X^P	Input local motion phase
X^S	Input pose features
X^V	Input control features

Neural network related symbols

$\mathcal{L}$	Loss
μ, σ	cluster centres (mean), scalars (standard deviation)
ϕ	activation function, basis function
Ψ	Output from the clustering layer (C-model)
Θ	Local motion phase
θ	Network parameters
C	Number of clusters / distribution factors
k	Number of experts in MoE model

w, b	Weights, biases
z	Embedding

1 Introduction

Animation is essentially moving pictures that are excellent at describing motions or phenomenon that changes over time. Computer animation refers to the computer-generated animation that brings the virtual characters to life in video games, cartoons and films. A character in this context refers to any animatable entity.

The earliest and the most common computer animation is interpolation-based animation, which uses predefined key-frames and let the computer generate transitions between them. Hand-crafting the key-frames has been the conventional method for creating such animations, but the production is time-consuming and not very scalable. It is less of a problem for cartoons or films, where there is only a fixed number of sequences that need to be crafted. However, in video games or other user-interactive applications, it is not very efficient to create animations for every possible situation in a dynamic environment.

The advancement in computer technologies has made real-time animation possible at a reasonable cost. It allows the animators to procedurally animate the characters that react to the user inputs and interact with the environment, resulting in dynamic, natural-looking and environment-aware motions. Deep learning-based motion synthesis systems are capable of learning from large and high-dimensional motion datasets, producing high quality and complex animations while having fast execution time and low memory footprints.

As with every other data-driven approach, these models require a large motion dataset for training, and the data are mostly motion captured animations which are expensive and time-consuming to produce. The interface of the models is dependent on the configuration of the character skeleton (rig), on which the motion data is recorded. Thus, the models can only produce animations for the particular rig.

Transfer learning techniques in the form of rig agnostic encoding methods are a possible solution to overcome this issue and enable the models to learn and generate motion irrespective of the rig configuration. This project investigates the possibility of three transfer learning techniques for objective-driven motion generation neural network (OMG), with the goal to propose a runtime framework for animating a character in real-time applications such as video games, and the framework can be easily adapted to other character rigs.

1.1 Background

Character rig is the skeleton of the virtual character that is animated in an animation sequence. It consists of a hierarchy of rotational joints that are linked together. A rig configuration includes the number of joints, degree of freedom, topology and proportions.

Key-frame animation is the traditional type of computer animation that is created by defining a start pose, an end-pose, and some intermediate poses that altogether describe the different states of the motion. *The character is animated by interpolating between the specified poses. The character pose of each keyframe in a sequence is either crafted by animators or recorded using motion-capture suites. The latter is referred to as motion-captured (mo-caps) animations, which is about recording and mapping motions from a real character to a virtual character, achieving realistic and smooth animations.*

The key-frame animations are good for controllability, reusability, reliability, artistic purposes and personalised features. Since they are static, it requires multiple variants, carefully designed transitions and blendings of different sequences for interactive applications such as computer games.

Procedural animation is the category for all the techniques that animate the characters programmatically, and these are generally goal-driven or physics-based. Unlike key-frame animations, procedural animations are dynamic, interactive and environment-aware. They can be used as secondary animations that act as a layer on top of the key-frame animations, for real-time adjustments such as foot-planting.

Full-body procedural animations are very complex and difficult to guarantee correctness and stability. Simple models might produce wrong behaviours or artefacts in unpredicted situations, and complex models are computationally demanding and not suited for real-time applications.

Deep learning-based procedural animation models aim to overcome these issues. The neural networks with their high capability and fast execution time, make them ideal for real-time applications. Some motion-synthesising models can blend segments from multiple sequences from the database to create new animations. Reinforcement learning-based models can learn to animate the characters in physically-based environments and perform various tasks. Some regression-based models can generate new sequences given the previous states and some control signals. All these models are bound to the character rig they are trained with and cannot directly be reused for new rigs.

Transfer learning refers to the techniques that transfer the knowledge in trained models to other models to improve their performance on their respective tasks. More formally, the definition involves two concepts: domains D and tasks T. Given a source domain and a learning task D_s, T_s and a target domain and target task D_t, T_t, transfer learning aims to improve the performance of the model on the target domain and task using knowledge learnt in the source domain and task.

1.2 Research question

The project is divided into 2 modules with the following research questions:

- **Objective-driven motion generation network (OMG)**
  (RQ1) Which of the two neural networks: Mixture-of-Experts (MoE) and Long-term-short-memory (LSTM), achieves better performance in terms of reconstruction error and adversarial error, with respect to the model complexity?
- **Transfer learning on OMG**
  (RQ2) Which of the three transfer learning techniques: feature encoding, feature clustering and feature reduction, is better at maximising the transferring quality and generation performance with limited training data, in terms of improvements in reconstruction error, positional-, rotational error and adversarial error, with respect to the model complexity?

1.3 Purpose

The purpose of this project is to study and investigate the possibilities of transfer learning on the most recent deep learning-based motion synthesis frameworks. To gain insights into the performance improvement when using various techniques and models with limited training and data.

1.4 Goal

The short-term goal of this project is to provide pioneering insights into how the transfer learning techniques can be applied to the motion synthesising networks for improving learning with limited training data. The long-term goal is to enable the training with motion data from character rigs of different configurations to improve the overall quality and enable the models to easily adapt to new character rigs.

1.5 Methodology

This project is problem-solving research that follows an experimental design with a deductive approach using quantitative methods. The literature study and the case study are conducted to gain a deeper understanding of the topics such as procedural animation, transfer learning and data-driven motion synthesis, to identify and formulate the problem.

A set of neural networks are selected and three transfer learning approaches are identified. The data involves multiple character rigs and motion data for each of the rigs. The rigs are obtained from various sources that represent real-world scenarios in the game industry. The motion data are created using an IK-solver in Unity for each of the rigs.

The neural networks are implemented in PyTorch and tested with the created motion data, to find a working architecture and tune the hyperparameters. A set of experiments are designed to test the performance of the models and the transfer learning techniques, based on the available data, computing power and time. The experiments are carried out on the local machine, the quantitative results are evaluated using statistical analysis methods for testing the null-hypothesis, to confirm the statistical significance of the improvements. The generated animations are replayed in Unity for observing the visual quality.

1.6 Benefits and Sustainability

By enabling the motion generation networks to learn and produce animations across the rigs, it reduces the amount of data and training required, saving a great amount of computational resources and time. It will also speed up the training of networks on new character rigs with limited data. Furthermore, it opens up new possibilities to train models for generating animations on characters, where no motion data are available, by using pre-trained models on similar domains.

1.7 Commissioned work

The project is sponsored and supported by EA Digital Illusion CE AB, a Swedish game studio that is behind the award-winning first-person shooter game series Battlefield and Star Wars: Battlefront. The studio provides the hardware and the mo-cap data for developing and training the networks. The supervisors from the company provide guidance, advice and help throughout the project

1.8 Delimitations

This project only focuses on bipedal, humanoid character rigs, though the results should apply to all other characters rigs such as quadruped regardless of the configuration since the networks are treating the joints as individual data points and not considering the dependencies between them.

The motion generation networks (MoGen) that are implemented and tested in this project are modified versions of the original models from the previous work, to cope with the created motion data and to fit the time scope. Ideally, to maximise the transferability of the results, the transfer learning techniques should also be tested on the original models with the original data.

All the motion data for training and testing are from the same category, namely reaching animations, where the character reaches forward with both hands to the targets. Diverse motion types are desirable for representing real-world scenarios, but due to the limited time and computing power, this project only focuses on transfer learning over different domains (rigs) and keep the task (motion type) fixed.

Although this project only measures the offline performance of OMG, the final aim to use the framework in real-time applications, hence the performance to complexity ratio is a critical factor for the evaluation.

1.9 Outline

2 Theoretical background

In this section, relevant information for understanding the various approaches for making computer animations, deep learning-based frameworks for generating animations and the role of transfer learning in the machine learning field is presented.

2.1 Computer animation

Conventional animation in the early 20th century was created by filming hand-drawn images. Walt Disney and his studio pushed the quality and the artistic value of animations to newer heights with technical innovations and the integration of sound. Technology drives innovation, with the advancement of computer technologies and digital tools, new possibilities for making animations have been unlocked. Three-dimensional (3D) animations started to appear at SIGGRAPH, and the pursue of realistic 3D animations started to gain popularity among the game developers and the filmmakers such as Lucasfilm (Pixar). The digitally modelled 3D characters can be animated by rotating the bones manually, or using projections on live actors or using physical rules and algorithms.

2.1.1 Interpolation-based animation

Interpolation-based animation is the most common type of computer animation, where the animator specifies a set of key poses (keyframes) that altogether define the motion. The character is animated by interpolating between the keyframes. Various interpolation techniques can be applied to create unique and stylish animations.

Many of the early computer animation systems were keyframe systems, that provide the animators with total control over the motion and the expected appearance. Handcrafting the keyframes required immense manual labour and knowledge to create natural and realistic animations.

2.1.2 Kinematic linkage

In computer animation, a character rig is an interactive interface for controlling the character skeleton, which the animators can manipulate to animate the character. It consists of a hierarchy of objects that represent the components of the rig, usually in the form of joints or bones. Kinematic linkage refers to dependencies between the objects. It describes the relation of the objects to their parents, which simulates the rigid connection between the biological joints in a human or a creature. An example of rig hierarchy in the tree structure is demonstrated in Figure 2.1

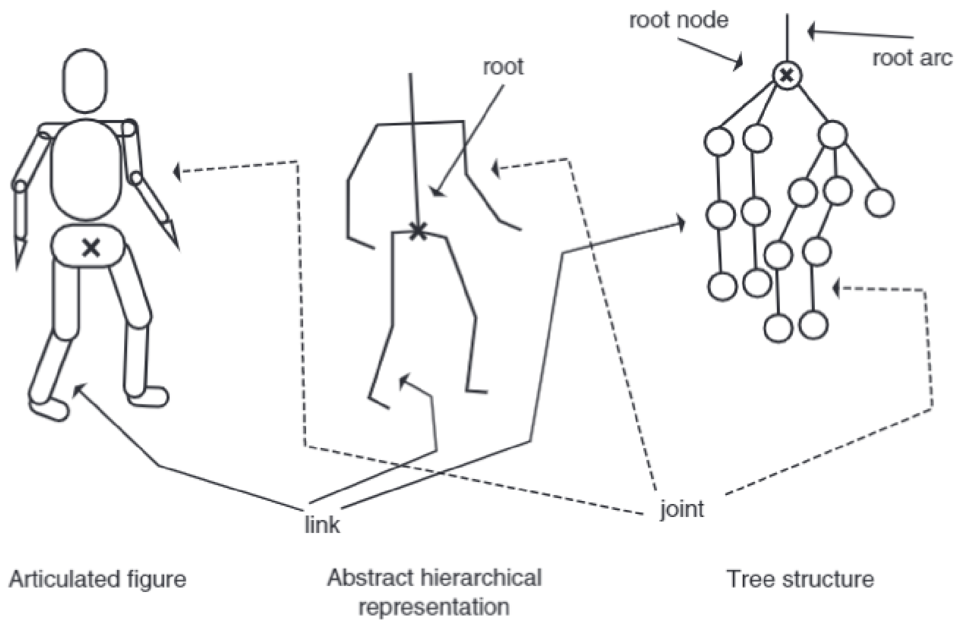

Figure 2.1: Example of a rig structure

The hierarchy can be represented by a tree structure, where the root node is the root of the rig that is often at the pelvis. The nodes are the joints and the links are the rigid connections between the joints. The leaf nodes are the end effectors. Each path from the root to a leaf node is a kinematic chain. Each node contains information both in the joint space, $\{\theta_x, \theta_y, \theta_z\}$ where $\theta_i \in \mathbb{R}$ is the joint angle around the i-axis, and the Euclidean space, $\{p \in \mathbb{R}^3, r \in \mathbb{R}^3, v \in \mathbb{R}^3\}$ where p is the position and r is the rotation in Euler angles and v is the linear velocity.

2.1.3 Forward- and inverse kinematics

Forward- and inverse kinematics are two mathematical processes that calculate the joint parameters of a kinematic chain, with respect to the kinematic linkage. Forward kinematic traverses through the hierarchy in a depth-first pattern, and propagates the changes of the parent nodes to end effectors, in the joint space. Conversely, inverse kinematics calculates the corresponding joint parameters of all parent nodes, from the end effectors' parameters in Euclidean space. These two processes are essential building blocks for robotics and digital animation tools.

The degree of freedom is the number of directions, in which a joint is allowed to rotate. Total degrees of freedom describe the complexity of the character rig. The higher the degrees of freedom the more possible solutions exist for an inverse kinematic problem.

Analytical solutions are possible for simple systems. For more complex systems, the Jacobian method can be used. By iteratively solving the Jacobian matrix of changes of end effector parameters, $\{p, r, v\}$ with respect to changes of the joint parameters.

$$\theta = [\theta_1, \theta_2, ..., \theta_n]^T$$

$$J = \begin{bmatrix} \frac{\partial p_x}{\partial \theta_1} & \frac{\partial p_x}{\partial \theta_2} & \cdots & \frac{\partial p_x}{\partial \theta_n} \\ \frac{\partial p_y}{\partial \theta_1} & \frac{\partial p_y}{\partial \theta_2} & \cdots & \frac{\partial p_y}{\partial \theta_n} \\ \cdots & \cdots & \cdots & \\ \frac{\partial r_z}{\partial \theta_1} & \frac{\partial r_z}{\partial \theta_2} & \cdots & \frac{\partial r_z}{\partial \theta_n} \end{bmatrix}$$

where θ is a vector of the changes to the joint parameters and n is the number of the joint in the system. Each term of the Jacobian describes the connection between the change of a specific joint to a specific change in the end effector. The desired configuration of the joint parameters can be obtained by solving the inverse of the Jacobian, that satisfies the specified parameters of the end effectors.

However, this numerical solution is not guaranteed to be natural-looking. Further control terms can be added to constrain the solution space or bias the solution towards some specific ideal joint angles. Another approach to solve inverse kinematic problems is by using cyclic coordinate descent. It chooses the best value for each joint sequentially, from the outermost inwards, that would get the end effectors to the target positions.

2.1.4 Motion capture (mo-cap

Motion capture techniques are widely used in film and game industries, for creating physically correct, natural-looking and detailed animations. While it is possible to achieve realistic and high-quality animations using a keyframe system with kinematic solvers, it is much faster to record real actors and project the recorded motions onto virtual characters for further processing.

In the game industry, for established game studios, it is the primary source for creating realistic and expressive animations. The mo-caps are further processed and cleaned using keyframe systems, and stored in a motion database. An animation system is usually a state machine, where each state corresponds to an animation clip in the database. When transiting from one state to another, a blending of the two corresponding clips is played.

Kinematic solvers can be applied on top of the animation system, to generate secondary animations that adjust the poses according to certain conditions, such as fitting the feet to the terrain.

2.2 Deep learning and transfer learning

Machine learning (ML) is about finding patterns in data and transform them to useful knowledge for tasks such as prediction and classification of new data. Deep learning (DL) is a subcategory in ML that utilises deep artificial neural networks (ANN) to efficiently capture complex patterns in large and high-dimensional datasets.

2.2.1 Basic concepts

A ML model can be expressed as a function $f(x;\theta)$ that takes the n-dimensional input vector $x \in \mathbb{R}^n$, with a set of parameters θ. Learning of a ML model refers to the optimisation of the function given some dataset $D = \{x_0, x_1, ..., x_M\}$.

There are generally three types of machine learning: *supervised learning*, *unsupervised learning* and *reinforcement learning*. The first type is the learning with labelled data $D = \{X, Y\}$, which is to find the optimal parameters θ^* such that the error between the $f(X;\theta^*)$ and Y is minimised, where Y is the labels(target outputs). The second type is learning that focuses on recognising the relationships between the data points, to extract latent information about the dataset. The third type is the learning of some optimal action policy that maximises some reward function, conditioning on the observations of some state.

A neural network consists of input- and output layer, and some hidden layers in-between them. The feed-forward layer is the most simple layer type in ANN. It performs the following operation:

$$h = \phi(w \cdot x + b)$$

where h is the layer output, w and b are the learnable network parameters: weights and biases. A visual exempel is displayed in Figure 2.2. $\phi(\cdot)$ is the activation function that is usually non-linear. For example, Rectified Linear Unit (ReLU) and Exponential Linear

Unit (ELU) are two activation functions:

$$\text{ReLU}(x) = max(0, x)$$

$$\text{ELU}(x) = \begin{cases} x, & x > 0 \\ \alpha(e^x - 1), & x <= 0 \end{cases}$$

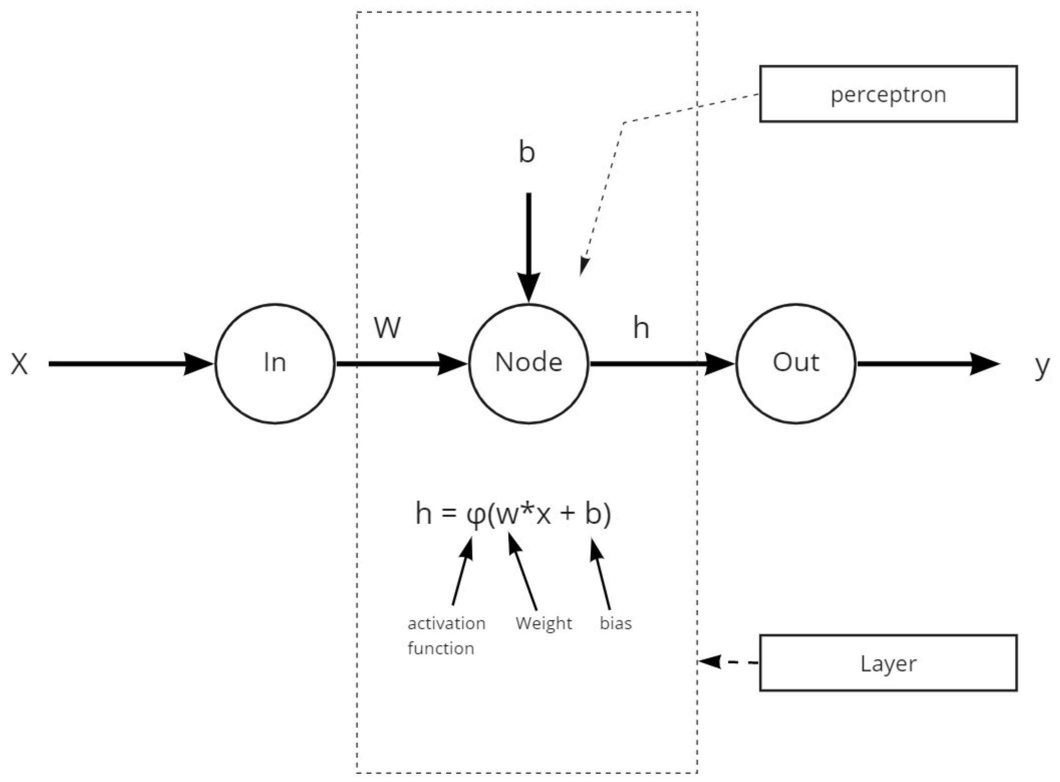

Figure 2.2: Example of network operation with a single layer and a single hidden node.

Optimisation of the network implies updating the parameters iteratively to reduce some loss function $\mathcal{L}(y_p, y_o)$, where $y_p \in \mathbb{R}^y$ is the predicted output and $y_o \in \mathbb{R}^y$ is the target output. A common loss function is mean squared error (MSE):

$$MSE(y_p, y_o) = \frac{1}{M}\sum_{i=0}^{M}(y_p^{(i)} - y_o^{(i)})^2$$

where the superscript denotes the ith sample in the dataset. The gradients of the loss function with respect to each of the parameters, { $\frac{\partial MSE}{\partial w}$, $\frac{\partial MSE}{\partial b}$ } are calculated and propagated back through the network to compute the gradients for each layer. The gradients are used to update the parameters at each layer:

$$\Delta w = -\eta \frac{\partial MSE}{\partial w}$$
$$\Delta b = -\eta \frac{\partial MSE}{\partial b}$$

where η is the learning rate that scales the update step. This optimisation process is called gradient descent and it is based on the MSE of the entire dataset, which can be burdensome for large datasets. A more computational efficient approach is called stochastic gradient descent (SGD), which processes the dataset in batches and computes gradients for each batch individually. One full iteration over all batches is called an epoch.

2.2.2 Training, validation and testing

To train a neural network, a common practice is to partition the dataset into 3 separate sets: training set, validation set and test set. The training set is used for optimising the network. The validation set is used for monitoring the learning of the network. The validation error indicates the performance of the network on unseen data, if this error increases after some epochs while the training error is steadily decreasing, it implies the network is overly adapted to the training data which results in worse generalisation performance. This behaviour is called overfitting. Conversely, if the optimisation fails to decrease the validation error, the network is said to be underfitting. The test set is used to measure the performance of the network after the training is finished.

2.2.3 Autoencoder (AE)

Autoencoder is an auto-associative deep learning model that learns efficient data encoding in an unsupervised manner. On the high level, the model is a function $f(x;\theta) = x$, that maps any input sample to the same input sample in the same space, as displayed in Figure 2.3. Internally, the model can be decomposed into two modules: an encoder (Enc$(\cdot)$) and a decoder (Dec$(\cdot)$). The encoder encodes the input sample to some latent representation (z) and the decoder decodes it back to the original input sample. A bottleneck can be formed by specifying the encoder to output a low dimensional z. In this way, the encoder is forced to learn to extract important features from the input samples. Thus, autoencoders can be used for feature extraction and dimensionality reduction. The latter is a common practice to handle the curse of dimensionality, which is a critical problem in ML. It refers to the high dimensionality of the data that makes learning difficult.

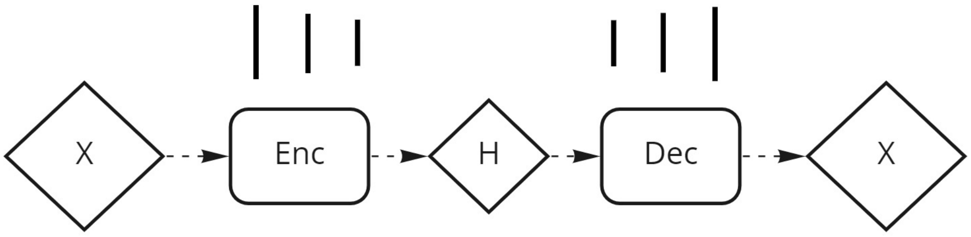

Figure 2.3: Visualisation of an autoencoder.

2.2.4 Autoregressive model (AR)

An autoregressive model is a type of deep learning model that predicts future values based on its own previous predictions. Autoregression is a regression of the variable against itself. An AR model of order p can be expressed as:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} + \varepsilon_t$$

where ε is a stochastic term, c is a constant term and ϕ_i is a parameter. AR models are suitable for time-series predictions and sequence data processing.

2.2.5 Transfer learning

Transfer learning has started gaining popularity in machine learning. It aims to improve the performance of target learners on target domains by transferring the knowledge in previously trained models. In this way, the amount of training data in the target domains, that is required to achieve comparable performance, can be reduced. Collecting a large amount of quality data is still an expensive and time-consuming process.

According to Zhuang et al:s comprehensive survey, the definition of transfer learning is given as the following:

> ***Definition 1.*** (Domain) *A domain D is composed of two parts, i.e. a feature space $\mathcal{X}$ and a marginal distribution $P(\mathcal{X})$. In other words, $D = \{\mathcal{X}, P(\mathcal{X})\}$. And the symbol $\mathcal{X}$ denotes an instance set, which is defined as $\mathcal{X} = \{x|x_i \in \mathcal{X}, i = 1, ..., n\}$*
>
> ***Definition 2.*** (Task) *A task $\mathcal{T}$ consists of a label space $\mathcal{Y}$ and a decision function f, i.e. $\mathcal{T} = \{\mathcal{Y}, f\}$.* The decision function f is an implicit one, which is expected to be learned from the sample data.
>
> ***Definition 3.*** (Transfer learning) *Given some/an observation(s) corresponding to $m^S \in \mathcal{N}^+$ source domain(s) and task(s) (i.e. \{ $D_{S_i}, \mathcal{T}_{S_i}|i = 1, ..., m^S$ \}), and some/an observation(s) about $m^T \in \mathcal{N}^+$ target domain(s) and task(s) (i.e. \{ $D_{T_i}, \mathcal{T}_{T_i}|i = 1, ..., m^T$ \}), transfer learning utilizes the knowledge implied in the source domain(s)*

to improve the performance of the learned decision functions f_{T_j} ($j = 1, ..., m^T$) on the target domain(s)

If $m^S = 1$, it is called single-source transfer learning, otherwise, it is called multi-source transfer learning. This project only focuses on the former type of transfer learning.

There are many strategies and techniques for transfer learning. The two primary strategies data transformation and model control. The former focuses on transforming the features in the target domains, such that the distribution differences between the source domains and the target domains are reduced. There are three types of feature transformation approaches feature augmentation, feature reduction and feature alignment. This project focuses on the feature reduction approach, which can be further divided into three categories: feature clustering, feature selection and feature encoding.

Feature clustering is about finding abstract feature representations of the original features. By utilising clustering techniques on the features or the instances, the relationship of the instances to the clusters can be used as input or augmented features for performing the task(s). Dai et al. proposed an unsupervised clustering approach that is called Self-Taught Clustering (STC). It assumes that the source domain and the target domain share the same feature clusters in their common feature space. STC co-clusters the source domain and the target domain instances simultaneously.

Feature selection is used to extract pivot features, which are the features that share the same behaviour in different domains.

Feature encoding is another method for feature extraction that produces an abstract representation of the instances, using autoencoders as described in Section 2.2.3.

2.3 Common neural network models

This subsection presents the relevant neural network models for this project.

2.3.1 Multi-layer perceptron (MLP)

Multi-layer perceptron (MLP), or feedfoward neural network, is the simplest and basic neural network model. It consists of feedfoward layers, where every node in a layer is connected to all the nodes in the next layer. A basic MLP network of L hidden layers that are parameterised by weights and biases can be expressed as the following:

$$f(x; W, B) = \phi_{L-1}(...(\phi_1(w_1 \cdot \phi_0(w_0 \cdot x + b_0) + b_1)))$$

where $\theta = (W = \{w_0, w_1, ..., w_{L-1}\}, B = \{b_0, b_1, ..., b_{L-1}\})$. MLP is generally good for regression tasks, or classification tasks by attaching a softmax layer to the output layer: $softmax(z) = \frac{e^{z_i}}{\sum_{j=0}^{K} e^{z_j}} \quad$ for $i = 1, ..., y \quad$ and $z \in \mathbb{R}^y$ and y is the output dimension.

2.3.2 Convolutional neural network (CNN)

Convolutional neural network (CNN) is commonly applied to computer vision-related tasks due to its space invariant property. At each layer, a convolution filter is applied to the input features and provide translation equivalent responses known as feature maps. CNN can extract locality-preserving latent features, which makes it robust at performing image-related tasks.

2.3.3 Radial-basis function (RBF)

Radial-basis function is a linear combination of C basis functions (kernels) $\phi_j(\cdot)$, where $j = 1, 2, ..., C$, that are only dependent on the radial distance from the respective centre μ_j, such that:

$$\text{RBF}(x) = \begin{pmatrix} \phi(||x - \mu_0||) \\ \phi(||x - \mu_1||) \\ ... \\ \phi(||x - \mu_C||) \end{pmatrix}^T$$

Common basis functions are:

- Gaussian basis function: $\phi(\alpha) = exp(-1 \cdot \alpha^2)$ where $\alpha = \frac{||x-\mu_j||}{\sigma}$ or $\alpha = \sigma||x - \mu_j||$.
- Quadratic function: $\phi(\alpha) = \alpha^2$.
- Spline function: $\phi(\alpha) = \alpha^2 \cdot ln(\alpha + 1)$

A RBF layer with Gaussian kernel is parametrised by $\theta = (\sigma, \mu)$ where $\sigma = \{\sigma_0, \sigma_1, ..., \sigma_C\}, \mu = \{\mu_0, \mu_1, ..., \mu_C\}$, can be expressed as the following:

$$f(x; \theta) = \sum_{i=1}^{C} \sigma_i \phi(||x - \mu_i||)$$

RBF is similar to MLP, but its capacity is limited by the number of kernels. The selection of the kernel centres is critical for RBF networks. K-means clustering based on Euclidean distance can be applied to the data points prior to training, to specify the kernel centres that no longer coincide with the training samples. Another approach is to let the centres be randomly initialised, by using a subset of data points as the centres or sampled from a normal distribution. The centres are part of the learnable parameters and get optimised during the training of the network.

2.3.4 Deep embedding clustering (DEC)

Deep embedding clustering (DEC) is an unsupervised deep learning-based clustering model proposed by Xie et al. It clusters the data points by simultaneously learning a set of C cluster centres $\{\mu_0, \mu_1, ..., \mu_C\}$ in the latent feature space $\mathbb{R}^k$, and the parameters θ of the MLP that maps the data point $x_i \in \mathbb{R}^n$ to $z_i \in \mathbb{R}^k$, where $k < n$ and $i = 1, 2, ..., M$.

DEC consists of two modules, an autoencoder based on MLP that learns the mapping function, and a clustering model that assigns the data points in the latent space to the C clusters. The clustering model performs a soft assignment that uses Students' t-distribution as the kernel function for measuring the similarity of the embedded points z_i to the respective cluster centres μ_j:

$$q_{ij} = \frac{(1 + ||z_i - \mu_j||^2/\alpha)^{-\frac{\alpha+1}{2}}}{\sum[(1 + ||z_i - \mu_j||^2/\alpha)^{-\frac{\alpha+1}{2}}]}, \quad \text{where } j = 1, 2, ..., C$$

DEC is optimised by minimising the reconstruction error produced by the autoencoder and the Kullback-Leibler (KL) divergence of the soft assignment distribution to the auxiliary target distribution:

$$p_{ij} = \frac{{q_{ij}}^2/f_i}{\sum_j {q_{ij}}^2/f_i}, \quad \text{where}$$
$$f_i = \sum_i q_{ij}$$

2.3.5 Variational autoencoder (VAE)

Variational autoencoder is a probabilistic variant of autoencoder. Instead of mapping data points to latent representations directly, the encoder produces a set of parameters that describe the latent distribution of the data points. The embeddings can be sampled from this distribution that can be used for the decoder to reconstruct the input sample.

Distribution of the encoder function can be described as:

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)}, \quad \text{where } p(x) = \int p(x|z)p(z)dz$$

where z is the embedding of the input sample x. $p(x)$ is a normalisation constant and usually an intractable distribution, making direct inference impossible. Instead, variational inference can be used to approximate $p(z|x)$ by construction another distribution $q(z|x)$ that is tractable. Optimising q implies minimising the KL divergence:

$$KL(q||p) = \sum_x q(x) log \frac{q(x)}{p(x)}$$

- $KL(q||p) \geq 0$ for all q, p.

- $KL(q||p) = 0$ if and only if $q = p$

The optimisation is achieved by raising the evidence to lower bound (ELBO):

$$\begin{aligned} &Let \quad p = p(z|x), \quad q = q(z|x), \quad Z = p(x) \\ &J(q) = \sum_x q\ log(\frac{q}{\tilde{p}}) \\ &J(q) = \sum_x q\ log(\frac{q}{p}) - log(Z) \\ &J(q) = KL(q||p) - log(Z) \end{aligned}$$

where $\tilde{p}$ is the unnormalised $p(z|x)$. Since $KL(q||p) \geq 0$, after rearranging the terms:

$$\begin{aligned} log(Z) = KL(q||p) - J(q) \geq -J(q) \\ log(Z) \geq \mathbb{E}_q[log(\tilde{p}) - log(q)] \end{aligned}$$

$-J(q)$ is the ELBO. By maximising this term, $KL(q||p)$ is "squeezed" between $-J(q)$ and $log(Z)$, since Z is a constant term. By mean-field theory, q can be decomposed into a set of fully factored distributions: $q = \prod_{i=1}^{C} q_i$, where each q_i is parameterised by θ_i. MLP are commonly used to produce a set of distribution parameters $f(x; \theta_{MLP}) = \theta_q, \quad \text{where } \theta_q = \{\theta_1, \theta_2, ..., \theta_k\}$. The latent representation $z_i \in \mathbb{R}^k$ can then be sampled from the joint distribution. A common practice is to set the target distribution $\tilde{p}$ to be normal distributed: $\tilde{p} \sim \mathcal{N}(0, 1)$, which also acts as regularisation on the learning.

Similarly, the distribution of the decoder can be expressed as $p(x|z)$ and is usually approximated using MLP. The loss function of VAE is formulated as the following:

$$\mathcal{L} = \mathbb{E}_q[log(p(x|z)] - \mathbb{E}_q[log(\tilde{p}) - log(q)]$$

where the first term is the reconstruction likelihood and the second term is approximation error to the target distribution.

2.3.6 Mixture-of-Experts (MoE)

Mixture-of-Experts (MoE) consists of a fully conditional mixture model where the blending coefficients, that is produced by a gating function, and the component densities, also called experts are conditional on some input variables. It is used in a variety of context including regression, classification and clustering.

The aim of regression is to recognise the relationship of an observed random variable $Y \in \mathbb{R}^y$ given a covariate sample $x \in \mathbb{R}^n$ by learning the conditional density function $p(Y|X = x)$. The univariate mixture of regression model, assumes that the observed pairs (x, y) are generated from Q regression functions that are governed by a hidden categorical random variable Z, indicating the component from which each observation is generated. A nonlinear regression model can be decomposed into a weighted sum of Q regression components:

$$f_k(y|x;\theta) = \sum_{i=1}^{k} \pi_i f_i(y|x, \theta_i)$$

where $\pi_i = P(Z = i)$ represents the non-negative blending coefficients such that $\sum_{i=1}^{K} \pi_i = 1$. In MoE, they are modelled as a function of some covariates, generally implemented using a softmax function:

$$\pi_i(x;\theta) = P(Z = i|x;\theta) = \frac{exp(\theta_i^T x)}{\sum_{i=1}^{k} exp(\theta_i^T x)}$$

2.3.7 Long short-term memory (LSTM)

Long short-term memory is a recurrent neural network that contains feedback connections from the output nodes back to the input nodes. It is widely used for sequential data processing, such as time series prediction, natural language processing and audio processing. An LSTM layer has a cell, an input gate, an output gate and a forget gate.

With these cells, it can memorise information from earlier inputs, making LSTM robust to longer sequence data. The operations in the LSTM layer are the following:

$$\begin{gathered}
f_t = \sigma_g(W_f x_t + U_f + h_{t-1} + b_f) \\
i_t = \sigma_g(W_i x_t + U_i + h_{t-1} + b_i) \\
o_t = \sigma_g(W_o x_t + U_o + h_{t-1} + b_o) \\
\tilde{c}_t = \sigma_c(W_c x_t + U_c + h_{t-1} + b_c) \\
c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \\
h_t = o_t \circ \sigma_h(c_t)
\end{gathered}$$

where

$x_t \in \mathbb{R}^d$:	input vector to the LSTM unit
$f_t \in \mathbb{R}^h$:	forget gate's activation vector
$i_t \in \mathbb{R}^h$:	input/update gate's activation vector
$o_t \in \mathbb{R}^h$:	output gate's activation vector
$h_t \in \mathbb{R}^h$:	hidden state vector
$\tilde{c}_t \in \mathbb{R}^h$:	cell input activation vector
$c_t \in \mathbb{R}^h$:	cell state vector
$W \in \mathbb{R}^{h\times d}, U \in \mathbb{R}^{h\times h}, b \in \mathbb{R}^h$:	weight matrices and bias vector

2.3.8 Least Square Generative Adversarial Network (LSGAN)

GAN is special type of generative models, that consists of two modules: a generator and a discriminator. The generator is trained to generates samples like the observed samples x_i in the datasets. The discriminator is trained to differentiate the generated samples from the real samples, adding extra error to the generator and forcing it learn better. Let $G(\cdot)$ be the generator and $D(\cdot)$ be the discriminator, the following loss terms are computed and used to optimise the models:

$$\begin{aligned}
\mathcal{L}_D =& \frac{1}{2}\mathbb{E}_{y\sim p_{data}(y)}[(D(y)-1)^2] + \frac{1}{2}\mathbb{E}_{x\sim p_x(x)}[D(G(x))^2] \\
\mathcal{L}_G =& \frac{1}{2}\mathbb{E}_{x\sim p_x(x)}[(D(G(x))-1)^2]
\end{aligned}$$

2.4 Related work

This section presents the related work in data-driven motion synthesis models.

2.4.1 Kernel-based approaches

Principal component analysis (PCA) has been successfully applied for reducing the dimensionality of full-body motion vectors, but the low dimensional latent space has issues capturing a wide range of movements. Kernel-based approaches have been proposed to overcome this issue. RBF and Gaussian process (GP) are two common kernel-based models for learning different types of locomotion.

2.4.2 Motion alignment models

A straightforward approach to synthesise motion using mo-caps is to align the motion sequences of the same motion type along the timeline and blend them with some blending coefficients computed by the model. Motion matching is a framework for synthesising motion by blending segments of different motion sequences in the database, that satisfy the constraints and the conditions. The optimal segments are obtained by computing the k-nearest neighbour at runtime. Min et al. proposed a model called motion graph++, where the motion sequences in the database are represented by functional PCA, and the optimal segments are obtained through maximum a posteriori (MAP).

2.4.3 Time series models

Time series models are the models that predict the future pose of the character based on the current and the past poses, conditioned on some control signals. Various neural network models have been proposed for this task, such as conditional Restricted Boltzmann Machine (cRBM), Gaussian Process (GP) and Encoder-Recurrent-Decoder model (ERD). Although these models are more robust and efficient than the kernel-based models, they suffer from drifting issues, where the generated motion comes off from the motion manifold and converges to a generic pose.

LSTM-based approaches are a natural choice for motion prediction, due to their capability of handling long sequence data. ERD is a model that incorporates an LSTM model with an autoencoder, that is capable of generating mo-cap quality animation, body pose labelling and body pose forecasting in videos. Harvey et al. proposed an LSTM-based model (TR-LSTM) for generating transitions between temporally-sparse keyframes, by introducing two novel additive embeddings: *time-to-arrival embedding* and *scheduled* *target noise embedding*. The LSTM models are difficult to tune and suffers heavily from bias-variance tradeoff, they are also less ideal for runtime generation tasks due to their low responsiveness caused by their high dimensional internal memory state.

Holden et al. proposed a novel neural network structure called Phase-Functioned neural network (PFNN), where the weights of the motion generation network (MoGen) is computed by Catmull-Rom spline function conditioned on a phase variable. The phase variable tells the progression of the motion and is defined by the foot contact pattern of bipedal locomotion. The introduction of the phase variable improved the quality of the generated motion that can adapt to different terrains.

Based on PFNN, Starke et al. proposed a framework called Mode-Adaptive neural network (MANN) for quadruped motion synthesis. MANN replaces the original phase function with a gating function, transforming the model into an MoE model. The gating function is conditioned on the feet velocities, action labels and target velocity. MANN is capable of generating believable quadruped locomotion with different gaits, adapting to different geometry and does not suffer from feet sliding issues.

Based on MANN, Starke et al. proposed two enhanced versions that are capable of generating character-scene interacting animations (Neural State Machine, NSM), and complex multi-contact character-object animations (MoE with local motion phase, LMP-MoE). The latter is capable of animating bipedal characters to play basketball, with various movement and interaction modes, by introducing a local motion phase variable. Unlike PFNN, where the locomotion is governed by one single global phase variable, LMP-MoE computes local motion phases for each key-bone (feet, hands and ball) based on the contact transitions. With the local motion phase variable, the network can generate both cyclic and acyclic motions where the body parts are moving at different and inconsistent velocities and frequencies.

Zinno et al. proposed a VAE-based framework for synthesising bipedal locomotion, referred to as Motion VAE (MVAE). It combines MoE with an autoregressive conditional VAE to predict the next pose given the current pose. Furthermore, the model can learn the distribution of next-pose predictions. This latent distribution is treated as the action space for their reinforcement learning-based animation controller, that combined with MVAE is capable of generating goal-driven motions.

2.4.4 Physically-based animation systems

Physically-based animation systems generate kinematic animations that obey the physical constraints or animations that are produced by applying forces (torques) to the character in a physically-based environment. A forward dynamic approach is proposed by Hodgins et al., that computes and applies joint torques to the body, to generate realistic motion. Peng et al. proposed a deep reinforcement learning framework called DeepMimic, that is capable of learning mo-cap data in a physically-based environment.

2.4.5 Evolutionary strategy-based IK solver

Starke et al. has proposed an evolutionary strategy-based IK solver that can solve fully constrained generic inverse kinematics with multiple end effectors and goal objectives. The solver uses a combination of evolutionary strategy optimisation technique and swarm optimisation, together with limited-memory-Broyden-Fletcher-Goldfarb-Shanno for gradient-based optimisation on inverse kinematic problems. The algorithm is fast, robust to avoid suboptimal extrema and capable of producing accurate solutions that satisfy biological constraints in real time. The solver is also developed into a plugin that is available on Unity Asset Store, which is used for creating the motion data for this research.

3 Methodology

In this section, the methodology and the processes are presented and explained in detail. The research design is presented in Section 3.1, which covers all scientific components such as the research approach, the research method and the research strategy.

The research is divided into five stages, starting from 1) understanding and formulating the problem; 2) selecting and designing the models and the transfer learning approaches; 3) collecting training data; 4) implementing and tuning the models and the approaches; 5) conducting the experiments and evaluating the results. The process is visualised in Figure 3.1

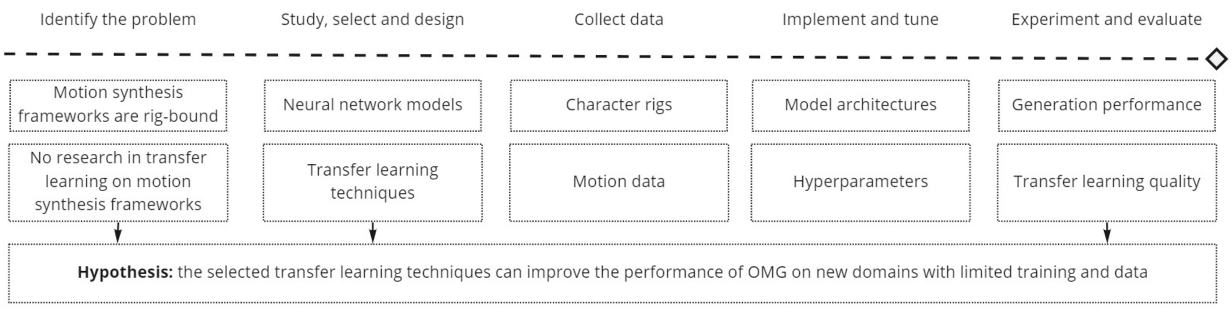
Identify the problem
Study, select and design
Collect data
Implement and tune
Experiment and evaluate
Motion synthesis frameworks are rig-bound
No research in transfer learning on motion synthesis frameworks
Neural network models
Transfer learning techniques
Character rigs
Motion data
Model architectures
Hyperparameters
Generation performance
Transfer learning quality
Hypothesis: the selected transfer learning techniques can improve the performance of OMG on new domains with limited training and data

Figure 3.1: The research process with the five stages.

3.1 Research overview

This is problem-solving research following the deductive approach and the experimental design and relies on qualitative and quantitative methods for evaluation of the results.

The purpose of this research is to study different neural network models for objective-driven motion generation, transfer learning techniques, and the dynamics between them, hence the experimental design and the deductive approach are a natural choice for constructing the research. Another dominant factor of this choice is the difficulty of proving and guarantee the correctness and the validity of the models in a theoretical manner, due to the complex internals of neural networks and the manifold nature of the datasets. Thus using the empirical results from the experiments that study the relationships between the variables and approaches, is a more time-efficient approach to demonstrate and validate the effect of the various models and techniques.

The experiments are designed to test the selected transfer learning techniques on the implemented objective-driven motion generation network (OMG). The results are evaluated both qualitatively by observing the generated motions, and quantitatively by measuring the errors and the accuracy of the models. The following qualities of the research are assured:

1. **Validity**, the quantitative results are validated using statistical analysis methods for confirming the statistical significance of the improvements.
2. **Reliability**, all the experiments are conducted on the same local machine, and by using a fixed random generator seed (2021), the randomness introduced in the system is controlled hence guaranteeing consistency throughout the experiments.
3. **Replicability**, all details of the implementations and the data generation method is presented in this thesis, making the research easily replicable for other researchers. Moreover, the source code of the project is available on GitHub.
4. **Ethics**, the implementation of the neural networks are based on published work and open-source projects, which are correctly referenced in the thesis. The character rigs are all available and free to use for educational and commercial purposes, except the rig received from DICE. The DICE rig is thus kept private and not included in the GitHub repository. All the motion data are original and created using the Bio-IK plugin in Unity.
5. **Transferability**, is the most important quality of this research. It describes whether the findings of this research applies to other relevant frameworks. This quality is discussed in Section .

3.2 Problem formulation

The problem is initially an extension to one of the future work directions described in the MANN paper. The direction is to extend MANN for motion retargeting tasks, making MANN capable of producing motions for quadrupeds of different size and morphology. Since the motion capture of the quadruped is a challenging task, and the motions for different body sizes might not be always available. Starke et al. also mentioned that it would be interesting to investigate feature transfer type of models to transfer the various gaits and movements from one animal to another animal. Collecting mo-caps of different animals with distinct gaits and styles is difficult and out of scope for this project.

Using MANN to animate other quadrupeds is only possible if the rigs share the same configuration, especially the number of joints since the input and the output layers of the model are dependent on the motion vectors, which in turn are dependent on the rig configuration. This limitation also applies to many other motion synthesis frameworks. The closest research is Neural Kinematic Network (NKN) for motion retargeting, by Villegas et al. It is based on regular Recurrent Neural Network (RNN) that projects the input pose from one character to other characters with different skeleton bone length, which is the only difference between the rigs. NKN is only for motion retargeting and cannot generate novel motions, and it does not work on rigs with a different number of joints.

The problem of this research is identified as investigating and finding methods for overcoming this limitation and making motion synthesis frameworks such as MANN function on other character rigs with different configurations. The problem falls under the transfer learning domain and is formulated as the following based on the definition of transfer learning described in Section 2.2.5:

- Given a **motion generation network** (MoGen) that is trained with **motion data** from **bipedal rig 1** (source domain D_S) to generate certain type of motion such as locomotion (source task T_S), what **transfer learning techniques** can be used to make MoGen produce same type of motion (target task $T_T = T_S$) on **bipedal rig 2** (target domain D_T) of different configuration, including different number of joints, with smaller training dataset?

3.3 Model selection

Ideally, the transfer learning techniques should be tested directly on the most recent motion synthesis frameworks such as LMP-MoE [2.4.3], MVAE [2.4.3] or TR-LSTM [2.4.3]. Fortunately, these models are well-documented and explained in the papers, and the code including the datasets are all available on GitHub, facilitating the replication of the models. However, since the focus is on transfer learning techniques, motion data of the same type on multiple rigs are required and these models are all conditioned on different control signals. In addition, for easier and consistent comparison between models and approaches, only the offline performance are considered, hence using joystick inputs as the control signal, as with LMP-MoE, is not an ideal choice to drive the pose prediction. MVAE uses a reinforcement learning-based controller, which would add unnecessary complexity and uncertainty to this project. TR-LSTM uses predefined keyframes as conditions for pose prediction, which suits the scope of this project but it would imply completely discard any possibility of using the model in a real-time environment in the future. Thus, a modified motion generation network is necessary.

3.3.1 Objective-driven Motion Generation network (OMG)

A framework that predicts the next pose based on the current pose and some objective function is proposed and referred to as Objective-driven motion generation network (OMG). It is essentially LMP-MoE with some cost as the condition, which is computed from some objective function. The objective function should return a quantity that needs to be minimised. For instance, it could be a function of the resultant force acting on the rigid body, making the framework to learn physics-based behaviours such as ragdoll physics.

For simplicity, the objective function is selected to be the Euclidean distance between the key joints (hands, feet) and their target positions and the angular difference to the target rotations. It is technically an MoE-based IK solver, approximating the behaviour of the Evolution strategy-based IK solver. By using temporally sparse targets, OMG can be effectively used as TR-LSTM. OMG can also be used in real-time applications by specifying the targets at runtime.

The local motion phase can be computed from contact patterns of the key joints and their target positions along the timeline. The main network (MoGen) can be either MoE or LSTM. Starke et al. did compare LMP-MoE with the LSTM counterpart and concluded that MoE performs better with fewer animation artefacts such as foot sliding and pose instability and achieved higher responsiveness. However, the models were tested in a real-time environment with user inputs as the control signal, hence it interesting to test whether the conclusion holds in the offline environment with fixed objectives as the control signal.

3.3.2 Transfer learning techniques

There are two main approaches in transfer learning, data transformation and model manipulation. Intuitively, the feature reduction techniques in the data transformation category are suitable candidates for this project. Since the target task remains the same as the source task, thus the MoGen should not be altered. Using feature reduction techniques to transform the data points to a more abstract representation in the shared feature space, reduces the complexity of the data and the differences between the different domains. The reduction techniques are implemented as the following:

1. **Feature encoding**, by integrating an autoencoder with MoGen, the pose data can be encoded into compact embeddings, which are fed to the MoGen. MoGen outputs the pose embeddings in the same latent feature space, which is then fed to the decoder to construct the pose vector in the original space.

2. **Feature clustering**, unsupervised clustering on the pose embeddings in the latent feature space can be performed by inserting an extra layer in between the autoencoder and MoGen. The relationship of the pose embeddings and the cluster centres can be used as inputs or auxiliary features to MoGen. The cluster centres can be considered as specific poses in the latent space and they should be selected to best capture the distribution of the pose domain. In this project, RBF, VAE and DEC are tested as the clustering layer.

   - RBF operates on the radial distance between the embeddings and the C clusters. It learns the centres $\{\mu_1, \mu_2, ..., \mu_C\}$ and the range of the centres $\{\sigma_1, \sigma_2, ..., \sigma_C\}$, and outputs a vector of radial distances to the respective centres given a pose embedding. An orthogonality loss term is added to regularise the clustering, promoting the layer to find orthogonal centres.

   - VAE learns the distribution of the latent pose domain, as factorised by C Gaussian distributions. The layer learns C set of Gaussian parameters $\{(\mu_1, \sigma_1), ..., (\mu_C, \sigma_C)\}$, and outputs a latent vector sampled from these distributions.

   - DEC is similar to RBF that learns a set of cluster centres but it uses Students' t-distribution for measuring similarity between the pose embeddings and the cluster centres. It outputs a vector of probabilities belonging to the respective cluster.

3. **Feature selection**, since the prominent issue of directly reusing the networks on other domains is the difference in the input and output dimensions, it can be avoided by simply using a fixed subset of features for all domains. More specifically, by using only six key joints (feet, hands, spine and head) to represent the pose, it can represent the poses from all domains in the feature vector of the same length, and also greatly reduces the dimensionality of the data. Kinematic solvers can be used to compute the parameters of other joints for the full-body pose, which adds extra

complexity. For simplicity, MoGen is trained to generate the next full-body pose given the pose embedding from the key joints.

For transfer learning, the autoencoder can be swapped with a new one that is pre-trained on the new domain, with the same layer sizes except for the input and output layers. When using feature clustering, the clustering layer is also transferred over to the new domain, to obtain the relationship of the pose embeddings from the new domain to the cluster centres that represent the previous domain.

3.4 Data collection

For experimenting with transfer learning, multiple bipedal character rigs and motion data on each of the rigs are required. The type of motion data needs to be the same for all rigs.

3.4.1 Character rigs

It requires a set of character rigs that share different configurations including the number of joints, the topology and the proportion. The more rigs and the more variation in the rigs, the better generalisation of the results. The experiments involve multiple repetitions of training various models on each rig, thus the experiment time is linearly proportional to the number of rigs. Due to the limited time, having a large number of rigs is infeasible. Instead, five rigs are collected from various sources that represent real-world scenarios in the game industry, which is the main application area of the framework.

1. (R1) A character rig from Adobe Mixamo. It is a clean and industry-standard character rig provided by Adobe for games and films. It is available for all creators and free to use in commercial projects.
2. (R2) A free bipedal monster rig from Unity Asset Store. It represents the type of rigs that would be used by indie studios with limited resources and funding.
3. (R3) The default character rig from Unity. It is the default rig that comes with Unity Engine and is accessible for all developers.
4. (R4) The character rig that is used in Neural State Machine (NSM) research project by Sebastian et al.
5. (R5) A character rig from DICE. It is currently being used in their ongoing projects. This rig represents the character rigs that are created by animators from established game studios to fit their specific needs, and it is used for mo-cap animations.

All the character rigs are bipedal humanoid rigs with different configurations.

3.4.2 Motion data

Creating motion data using motion capture is not feasible due to limited resources and time. Since OMG is objective-driven, using IK-solver to procedurally animate the characters to create motion data is a reasonable choice. This method is time-efficient and scalable, and it ensures full control over the generated motion data for all the rigs.

The motion sequences are created in Unity using the Bio-IK plugin by Sebastian et al. The positional and rotational objectives are used to drive the end effectors (hands) towards the target positions and the target rotations. All the clips have the length of 300 frames (5 seconds at 60 frames per second). The following clips are created for each rig:

- For each clip, the targets for both hands are spawned around the character and it reaches the hands towards the targets. When the distance between a hand and its target is less than 0.1m, it is considered the contact state. The hand stays at that position for 5 frames, before the target despawns.

  The process repeat (1 / 2 / 3) times for each clip, in which the targets are discreetly spawned on the surface of a cylinder with the root of the rig as the centre. The height of the cylinder is 0.7 (Unity) meter and the radius is (0.35m / 0.7m). The cylinder is divided into 5 levels with 8 spawn points on the perimeter of each level, evenly distributed with $\frac{2\pi}{8}$ degrees between them.

  For a single repetition case, 40 clips are created with the targets spawning on every spawn point (with/without) random rotation. After the targets are despawned, they immediately respawn on the initial positions, guiding the hands back to the default positions and orientations.

  For multiple repetition case, there are (2) spawn modes. In both modes, 40 clips are created where the spawn points are randomly selected, but in mode 1, adjacent spawn points for the targets are sampled and in mode 2, the opposite spawn points are selected such that the angle difference between them is π.

  In total, there are 960 sequences created, which corresponds to 288 000 frames (80 minutes) of motion data.

The sequences are generated using Bio-IK with the following parameters:

Generations	3
Individuals	120
Elites	2
Motion type	Realistic
Maximum velocity	5
Maximum acceleration	5

Each frame of the created motion sequences contains the parameters of the joints, the parameters of the targets and the respective cost in each frame.

3.5 Implementation and tuning of the models

The models are implemented in Python using Pytorch and Pytorch-Lightning. The created motion data are exported from Unity as JSON files, which are parsed and extracted to Numpy arrays and stored as bzip2-compressed binary files. The source code of MVAE, NSM and MANN is available on GitHub, which facilitated the implementation of the models.

The implementations are tested with a small subset of the dataset, to ensure the correctness of the implementation, that is, the reconstruction errors are optimised during the training. The hyperparameters such as the number of layers, the layer sizes and the learning rates are tuned using Ray Tune with ASHA scheduler and a grid search algorithm.

The generated animation is visualised in Unity to further verify that the generation process functions correctly, meaning that the hands of the characters are reaching towards targets.

3.6 Experimentation and evaluation

The experiments are designed to test the effect of the transfer learning techniques, more specifically, how much the performance is improved by warm starting the OMG on the domains R2, R3, R4 and R5, by loading the trained parameters of OMG on R1. All the experiments are run on the local machine with the following specifications:

- OS: Arch Linux with the linux kernel version 5.12.5
- CPU: AMD Ryzen 9 3900 @ 4.2 GHz
- GPU: Nvidia GTX 1080 Ti

- RAM: 32 GB

3.6.1 Evaluation methods

The collected performance data from the experiments are compared and the differences are computed. The statistical significance of the improvements is calculated and confirmed using p-values and confidence interval. The null hypothesis is tested using the Mann-Whitney-U test.

The best performing models are used to generate animation samples that are replayed in Unity, for observation of the visual quality.

4 System overview

This section presents the specifications of the data and the models.

4.1 Data

The motion data that is used for training and testing the models and the transfer learning technique, is the reaching animation clips that are generated using an IK-solver in Unity, for each of the five character rigs. In each clip, the character reaches forward with both hands to the targets, minimising the distance to the target position and the target rotation. The specification of the generation procedures of motion data is the following:

Spawn points	Number of repetitions	Random rotation	Spawn radius (m)	Spawn mode
40	1	False/True	0.7/0.35	-
40	2	False/True	0.7/0.35	adj./opp.
40	3	False/True	0.7/0.35	adj./opp.

where adj. denotes the spawn mode that the adjacent spawn points are selected, and opp. denotes the mode that the opposite spawn points are selected. In total, there are 960 motion clips in the dataset.

4.1.1 Character rig specification

The specification of the five character rigs are the following:

Rig	From	Joints	DOF	Average bone length (m)
R1	Adobe Mixamo	31	71	0.175
R2	Unity Asset Store	31	71	0.191
R3	Unity	29	65	0.189
R4	NSM repository	24	60	0.210
R5	DICE	32	74	0.169

There is a dataset for each rig, resulting in 5 datasets of 4800 clips in total.

4.1.2 Data features

The models are primarily based on LMP-MoE and MVAE, thus the data features for working with models are similarly formatted. The following features are included in the dataset:

- **Character pose** $X_i^S = \{p_i, r_i, v_i\}$ represents the pose of the character with J number of joints at the current frame i. $p_i \in \mathbb{R}^{3J}$ is the joint positions; $r_i \in \mathbb{R}^{6J}$ is the joint rotations in the form of a pair of the Cartesian forward and up vectors; $v_i \in \mathbb{R}^{3J}$ is the joint linear velocity.

- **Control variables** $X_i^V = \{T_i^p, T_i^r, C_i^p, C_i^r\}$ represents the control signals used to drive the prediction of the future poses. $T_i^p \in \mathbb{R}^{3K}$ is the target positions, where K is the number of key joints; $T_i^r \in \mathbb{R}^{6K}$ is the target rotation; $Tc_i^p \in \mathbb{R}^{3K}$ is the position cost for each key joint; $Tc_i^r \in \mathbb{R}^{3K}$ is the rotation cost.

- **Local motion phase** $X_i^P = \Theta_i \in \mathbb{R}^{2K}$ is the local motion phase vector for each key joint. The details of the phase vector is described in Section 4.1.3.

All the features are recorded in the rig root space.

4.1.3 Local motion phase

The local motion phase describes the progression of the motion for each joint individually, based on the contact pattern with the target. It contains information about the timing and the speed of the movement. The feature is computed in the same way as originally described in LMP-MoE paper but without the optimisation process, due to the original procedure failed to correctly produce desired values for the motion data.

Let $G(t)$ be the block function of a key joint, where $G(t) = 1$ if contact and $G(t) = 0$ otherwise. $G(t)$ is first normalised in a sliding window $W = 30$ frames centered at the frame t:

$$G_{norm}(t) = \frac{G(t) - \mu_{G_W}}{\sigma_{G_W}} \tag{1}$$

where $G_{norm}(\cdot)$ is the normalised block function, μ_{G_W}, σ_{G_W} are the mean and the standard deviation of $G(t)$ in the window G_W. The sliding window is padded with edge values and if $\sigma_{G_W} = 0$, then it is set to 1. The Butterworth low-pass filter $\mathcal{B}(\cdot)$ of order 3 with the threshold = 0.1 is applied to th e block function.

$$G_{\mathcal{B}} = \mathcal{B}(G_{norm}) \tag{2}$$

The local motion phase vector is constructed as the following:

$$\Theta = \begin{pmatrix} \Delta_{sin} \cdot sin(G_{\mathcal{B}}) \\ \Delta_{cos} \cdot cos(G_{\mathcal{B}}) \end{pmatrix} \tag{3}$$

where $\Delta_{sin} = sin(G_{\mathcal{B}}(t)) - sin(G_{\mathcal{B}}(t-1)), \Delta_{cos} = cos(G_{\mathcal{B}}(t)) - cos(G_{\mathcal{B}}(t-1))$

4.1.4 Objectives

The objectives are the target position and the target rotation. The objective cost $\{C^p, C^r\}$ for a key joint j, in the form of vectors, are the conditioning features that guide the prediction of future state variables of the joint. C^p is the Euclidean distance from the joint position to the target position with respect to the root coordinates. C^r are the angular distance between the joint and the target {forward, up} vectors.

$$C^p = T^p(j) - p(j) \tag{4}$$

$$C^r = \frac{1}{\pi} cos^{-1}(\frac{T^r(j) \cdot r(j)}{||T^r(j)|| \cdot ||r(j)||}) \tag{5}$$

4.2 Models

The following neural network models are implemented:

1. **Autoencoder [2.2.3], AE**$(\cdot)$ with an encoder Enc$(\cdot)$ and a decoderDec$(\cdot)$ with the same architecture, where both are implemented as three-layer MLP [2.3.1] with following network operation:

$$\text{Enc}(x; W, B) = \text{Dec}(x, W, B) = w_3\text{ELU}(w_2\text{ELU}(w_1 \cdot x + b_1) + b_2) + b_3 \tag{6}$$

   where ELU is the Exponential linear unit activation function; $W = \{w_1, w_2, w_3\}$ and $B = \{b_1, b_2, b_3\}$ are the learnable parameters of the network.

2. **LSGAN discriminator [2.3.8], D**$(\cdot)$, a temporal convolutional discriminator that learns the distribution of the pose data from the true dataset and provides an adversarial loss $\mathcal{L}_{\text{adv}}$ that tells the probability of generated pose is being sampled

from the real pose distribution.

$$L1 = \text{BatchNorm}(\text{MaxPool}(\text{Conv}(x, w_1, b_1))) \tag{7}$$

$$L2 = \text{BatchNorm}(\text{MaxPool}(\text{Conv}(L1, w_2, b_2))) \tag{8}$$

$$L3 = w_3(\text{Flatten}(L2)) + b_3 \tag{9}$$

$$D(x; W, B) = L3(L2(L1(x))) \tag{10}$$

where BatchNorm(·) is a batch normalisation layer, MaxPool(·) is a max-pooling layer, Conv(·) is a 2D convolutional layer and Flatten$(\cdot)$ is the flatten operation.

3. **RBF clustering layer [2.3.3], RBF**$(\cdot)$ with Gaussian basis function $\phi(\cdot)$ for clustering the pose embeddings.

$$\text{RBF}(x; \mu, \sigma) = \phi(\sigma||x - \mu||) \tag{11}$$

With a orthogonality loss $\mathcal{L}_{ortho}$ as the regularisation term:

$$\mathcal{L}_{ortho} = 0.1 \cdot \mathbb{E}(|\mu\mu^T - I|_1) \tag{12}$$

4. **VAE layer [2.3.5], VAE**$(\cdot)$ with two modules $\{\mu(\cdot), \sigma(\cdot)\}$ that outputs the parameters of C Gaussian distributions. The KL-divergence of the output distributions to the unit Gaussian distribution is used as a regularisation term.

$$\text{VAE}(x; W, B) = \{\mu = w_1 \cdot x + b_1, \sigma = w_2 \cdot x + b_2\} \tag{13}$$

The reparameterisation trick is implemented to sample the latent vector from the computed distributions:

$$z = \mu + \sigma \cdot \varepsilon, \quad \text{where } \varepsilon \sim \mathcal{N}(0, 1) \tag{14}$$

With a KL-loss $\mathcal{L}_{\text{KL}}$ of the C Gaussian distributions to the normal distribution $\mathcal{N}(0, 1)$, as the regularisation term:

$$\mathcal{L}_{\text{KL}} = \mathbb{E}[-\frac{1}{2}\sum(1 + \sigma - \mu^2 - e^{\sigma})] \tag{15}$$

5. **DEC layer [2.3.4], DEC**$(\cdot)$ that outputs a probability vector of the given input vector belonging to the respective cluster centres. The KL-divergence of the output

distribution to the auxiliary distribution is used as a regularisation term.

$$\text{DEC}(x;\mu) = \frac{(1+||x-\mu||^2)^{-\frac{1}{2}}}{\sum[(1+||x-\mu||^2)^{-\frac{1}{2}}]} \tag{16}$$

With a KL-loss $\mathcal{L}_{\text{KL}} = \text{KL}(q||p)$ of the output distribution q to the auxiliary distribution p as described in Section 2.3.4.

6. **MoE network [2.3.6] MoE**$(\cdot)$ with k experts is used for motion generation with a gating network $\Omega(\cdot)$ and a main network $N(\cdot)$. Both networks are implemented as three-layer MLP, where the main network has k sets of parameters which are blended using the coefficients from the gating network. The gating network is conditioned on the local motion phase variable Θ_i and produces the blending coefficients for blending the parameters on each layer. The main network takes pose embedding z_i and cost embedding c_i as input $x = [z_i, c_i]$, and outputs local motion phase update $\Delta\Theta$, next pose embedding z_{i+1} and next cost c_{i+1}, where i denotes the ith frame.

$$\Omega(\Theta; W, B) = U = w_3\text{ELU}(w_2\text{ELU}(w_1 \cdot x + b_1) + b_2) + b_3 \tag{17}$$

$$N(x; W, B, U) = w_3\text{ELU}(w_2\text{ELU}(w_1 \cdot x + b_1) + b_2) + b_3 \tag{18}$$

$$\text{where } w_l = \sum_{e=1}^{K} W_i^{(e)} \cdot U^{(e)}, \quad l = 1, 2, 3 \tag{19}$$

where e denotes the e-th expert.

7. **LSTM network [2.3.7] LSTM**$(\cdot)$ is a network with 4 LSTM-layers. It a many-to-many type of recurrent architecture that takes local motion phase Θ_i, pose embedding z_i and cost embedding c_i as input and outputs local motion phase update $\Delta\Theta$, next pose embedding z_{i+1} and next cost c_{i+1}. The operation details are provided in Section 2.3.7.

8. **Objective-driven motion generation network OMG**$(\cdot)$ is the complete framework for generating motion given the objectives. It consists of four modules: AE$(\cdot)$ is the autoencoder for encoding and decoding the pose data; C$(\cdot)$ is the clustering layer; CostEnc$(\cdot)$ is a three-layer MLP for encoding the cost data; MoGen$(\cdot)$ is the main generation network (either MoE or LSTM). The following are three variants of OMG:

FE-OMG$(x;\theta)$ is a framework consisting of only AE and MoGen. It implements the feature encoding technique. The pose encoding z is directly used as input to MoGen. The architecture is visualised in Figure 4.1.

$$
\begin{aligned}
X_i^P, X_i^S, X_i^V =& \text{slice}(X_i) \\
z_i =& \text{Enc}(X_i^S) \\
c =& \text{CostEnc}(X_i^V) \\
Y_{i+1} =& \text{MoGen}(X_i^P, z_i \oplus c) \\
\Delta\Theta, z_{i+1}, \{C_{i+1}^p, C_{i+1}^r\} =& \text{slice}(Y_{i+1}) \\
Y_{i+1}^P =& \text{update}(X_i^P, \Delta\Theta) \\
Y_{i+1}^S =& \text{Dec}(z_{i+1}) \\
Y_{i+1}^V =& \{T_i^p, T_i^r, C_{i+1}^p, C_{i+1}^r\} \\
Y_{i+1} =& Y_{i+1}^P \oplus Y_{i+1}^S \oplus Y_{i+1}^V
\end{aligned}
\tag{20}
$$

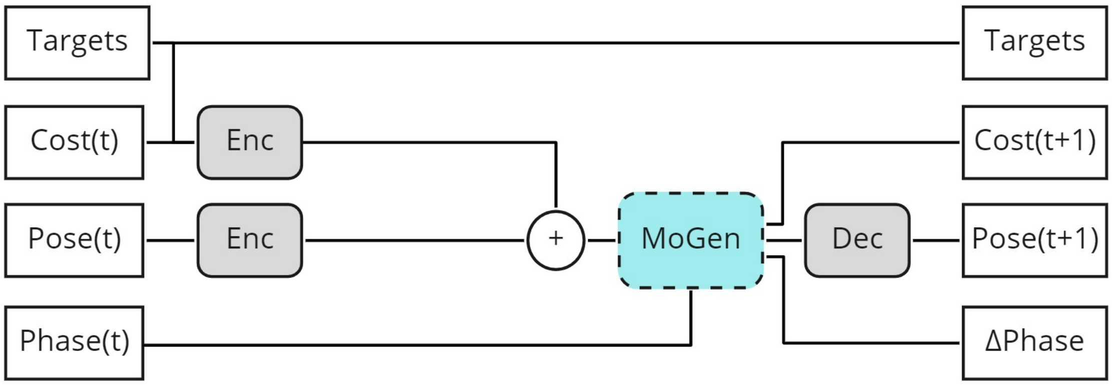

Figure 4.1: Architecture diagram of FE-OMG, where Targets is $\{T_t^p, T_t^r\}$, Cost(t) is $\{C_t^p, C_t^r\}$, Pose(t) = X_t^S, Phase(t) is X_i^P and t is the frame index. The addition symbol denotes concatenate operation. Enc/Dec denotes a three-layer MLP encoder/decoder and MoGen is the motion generation network (MoE / LSTM)

FC-IN-OMG$(x;\theta)$ is a framework that uses a clustering layer (either RBF or VAE or DEC) to obtain the cluster information Ψ of the pose embeddings, which is used as input to MoGen. Figure 4.2 displays the architecture diagram.

$$\begin{aligned}
X_i^P, X_i^S, X_i^V =& \text{slice}(X_i) \\
z_i =& \text{Enc}(X_i^S) \\
\Psi =& \text{C}(z_i) \\
c =& \text{CostEnc}(X_i^V) \\
Y_{i+1} =& \text{MoGen}(X_i^P, \Psi \oplus c) \\
\Delta\Theta, z_{i+1}, \{C_{i+1}^p, C_{i+1}^r\} =& \text{slice}(Y_{i+1}) \\
Y_{i+1}^P =& \text{update}(X_i^P, \Delta\Theta) \\
Y_{i+1}^S =& \text{Dec}(z_{i+1}) \\
Y_{i+1}^V =& \{T_i^p, T_i^r, C_{i+1}^p, C_{i+1}^r\} \\
Y_{i+1} =& Y_{i+1}^p \oplus Y_{i+1}^S \oplus Y_{i+1}^V
\end{aligned} \tag{21}$$

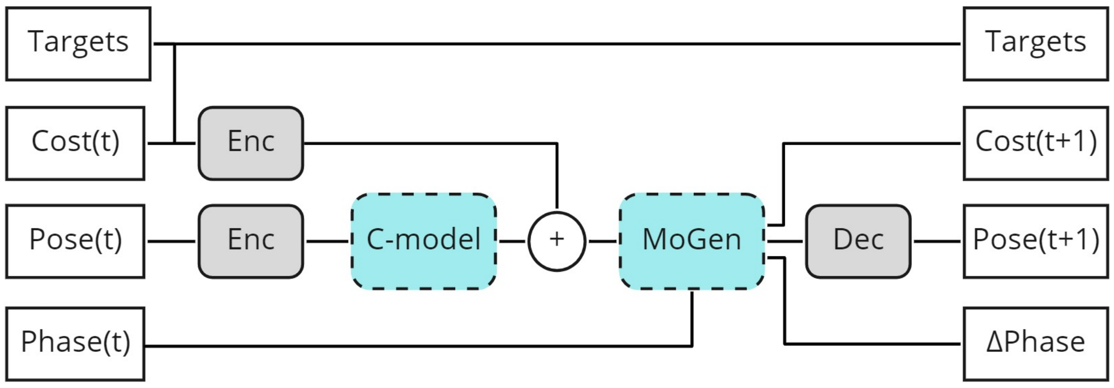

Figure 4.2: Architecture diagram of FC-IN-OMG, where Targets is $\{T_t^p, T_t^r\}$, Cost(t) is $\{C_t^p, C_t^r\}$, Pose(t) = X_t^S, Phase(t) is X_i^P. The addition symbol denotes concatenate operation. Enc/Dec denotes a three-layer MLP encoder/decoder, C-model denotes the clustering layer (RBF/VAE/DEC) and MoGen is the motion generation network (MoE / LSTM)

FC-CAT-OMG$(x; \theta)$ is similar to FC-IN-OMG, but using Ψ as auxiliary features, that are concatenated with the pose embedding z before feeding to MoGen. The network is visualised in Figure 4.3.

$$
\begin{aligned}
X_i^P, X_i^S, X_i^V =& \text{slice}(X_i) \\
z_i =& \text{Enc}(X_i^S) \\
\Psi =& \text{C}(z_i) \\
c =& \text{CostEnc}(X_i^V) \\
Y_{i+1} =& \text{MoGen}(X_i^P, z_i \oplus \Psi \oplus c) \\
\Delta\Theta, z_{i+1}, \{C_{i+1}^p, C_{i+1}^r\} =& \text{slice}(Y_{i+1}) \\
Y_{i+1}^P =& \text{update}(X_i^P, \Delta\Theta) \\
Y_{i+1}^S =& \text{Dec}(z_{i+1}) \\
Y_{i+1}^V =& \{T_i^p, T_i^r, C_{i+1}^p, C_{i+1}^r\} \\
Y_{i+1} =& Y_{i+1}^P \oplus Y_{i+1}^S \oplus Y_{i+1}^V
\end{aligned} \tag{22}
$$

where $\oplus$ denotes concatenate operation, slice$(\cdot)$ is the slicing operation splitting the vector into components on axis 1 (columns). update$(\cdot)$ is the local motion phase update function:

$$
\text{update}(X_i^p, \Delta\Theta) = 0.9 \cdot X_i^p + 0.1 \cdot \Delta\Theta \tag{23}
$$

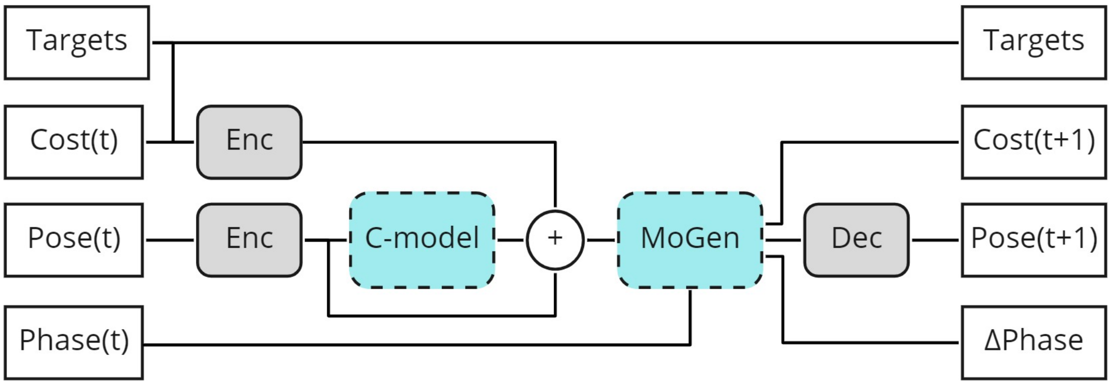

Figure 4.3: Architecture diagram of FC-CAT-OMG, where Targets is $\{T_t^p, T_t^r\}$, Cost(t) is $\{C_t^p, C_t^r\}$, Pose(t) = X_t^S, Phase(t) is X_i^P. The addition symbol denotes concatenate operation. Enc/Dec denotes a three-layer MLP encoder/decoder, C-model denotes the clustering layer (RBF/VAE/DEC) and MoGen is the motion generation network (MoE / LSTM)

4.2.1 Model configurations

The following configuration of the respective network is used for the experiments.

Model	Input dimension	Output dimension	hidden dimension/kernel size
Enc	IN	256	256
Dec	256	OUT	256
D	256	1	3×3
RBF	256	128	-
DEC	256	128	-
VAE	256	128,128	-
MoE	8, 256/384	8,256,24	256
LSTM	8, 256/384	8,256,24	256
CostEnc	60	128	128

where IN and OUT are the dimension of pose vector.

The special parameters of certain models are summarised in the table below:

Model	Parameter	Value
RBF	C	128
VAE	C	128
DEC	C	128
MoE	k	4
MoE	Gate size	128
LSTM	Layers	4

4.3 Network training

An input vector of OMG is $X_i = \{X_i^P, X_i^S, X_i^V\}$, where each item is described in Section 4.1.2. An input sample (clip) is a matrix of shape $299 \times n$, where n is the length of X_i.

$$X = \begin{pmatrix} X_0 \\ X_1 \\ ... \\ X_{299} \end{pmatrix}$$

For training, the input samples are stacked into a three-dimensional matrix for each batch with batch size=32. The matrix has the shape $32 \times 299 \times n$. The matrix is later decomposed into sequences of length 13, forming a new matrix with shape $736 \times 13 \times n$. The sequences are processed frame-wise, where the final input matrix to the network at each training step has the shape $736 \times 1\ timen$.

The output vector is $Y_i = \{Y_i^P, Y_i^S, Y_i^V\}$, sharing the same length as X_i. With teacher forced learning enabled, a uniform autoregression probability is used to determine whether the output vector is fed back to the network as input for the next frame.

All modules are trained in end-to-end fashion, with AdamW optimiser with weight decay rate 0.01 and $\beta = \{0.9, 0.999\}$. The learning rate is set to $1.0 \cdot 10^{-4}$ with the decay rate 0.9 every 80 training steps. Teacher forcing technique is applied, with a uniform probability, the output of OMG is fed back to the network as input for the next frame (autoregression). The autoregression probability is set to 0 at the beginning and is gradually incremented to 1.

The following loss terms are computed during training:

$$\begin{aligned}
\mathcal{L}_{\text{recon}} &= \text{MSE}(Y_i, X_i) \\
\mathcal{L}_r &= \text{MSE}(y_i^r, x_i^r) \\
\mathcal{L}_{\text{adv}} &= \text{D}(X_i^S) \\
\mathcal{L}_{\text{KL}} &= \text{KL}(X_i^S) \quad \text{if VAE or DEC is used} \\
\mathcal{L}_{\text{ortho}} &= 0.1 \cdot \mathbb{E}(|\mu\mu^T - I|_1) \quad \text{if RBF is used}
\end{aligned}$$

where only $\mathcal{L}_{\text{recon}} + \mathcal{L}_{\text{KL}} + \mathcal{L}_{\text{ortho}}$ is used for optimising OMG.

4.4 Experiments

The following experiments are conducted:

1. **(E0)** Train the autoencoder and its LSGAN discriminator on all domains (R1-R5) for 200 epochs.

   Expected results: a set of trained autoencoders $\{\text{AE}_{R1}, \text{AE}_{R2}, ..., \text{AE}_{R5}\}$, the reference performance data of the autoencoders and the reference adversarial losses from the discriminator.

2. **(E1)** Train OMGs (FE-OMG, FC-IN-OMG, FC-CAT-OMG) on all domains (R1-R5) for 51 epochs, with the autoregression probability starting at 0 and incremented with 0.5 every 20 epochs. 80% of the dataset is used as training data, 10% is used for validation and 20% is used for testing (including the validation set).

   Expected results: a set of trained OMGs with the reference performance data on all the domains. Demonstrating what kind of performance is expected if trained with the full dataset for each model.

3. **(E2)** Train the OMGs on R2-R5 respectively for 31 epochs, with the autoregression probability incremented with 0.5 every 10 epochs. Only 16% of the dataset (20% of the training set) is used for training, 10% is used for validation and 84% is used for testing (including the validation set and the remainder of the training set).

   Expected results: the cold-start performance data of the models on all domains (R2-R5) with limited training and data.

4. **(E3)** Same procedure as E2 but warm start the models by loading the parameters from the reference models OMG$_{R1}$. The clustering layer (if any) and the MoGen is fixed and not optimised during the training. Only the autoencoder in OMGs is trained.

   Expected results: the warm-start performance data of the models on all domains (R2-R5) with the knowledge in the pre-trained reference models on R1 transferred. The clustering layer and the MoGen are frozen to better demonstrate the effect of the transfer learning techniques.

5. **(E4)** Same procedure as E3 but allow the clustering layer and the MoGen to be optimised during the training.

   Expected results: the performance data showing how much the performance can be improved by further training the models on the data from the new domain (R2-R5).

6. **(E5)** Perform the E1-E4 on a reduced feature set (six joints instead of the full-body pose).

   Expected results: the performance data of using the feature selection technique + upsampling with MoGen.

The variables that are experimented in each experiment, are the following:

- **The transfer learning techniques:**
  - Feature encoding: (FE-OMG).
  - Feature clustering: (FC-IN-OMG), (FC-CAT-OMG)
  - Feature selection: using reduced feature set instead.
- **MoGen**: MoE, LSTM
- **Clustering layer**: RBF, VAE, DEC

4.4.1 Metrics

The following metrics are measured for quantifying the performance of the models:

- **Reconstruction error,** $\mathcal{L}_{\text{recon}}$, the mean squared error of the generated output against the target output.

$$\mathcal{L}_{\text{recon}} = \text{MSE}(Y_i, X_i) \tag{24}$$

- **Positional error,** $\mathcal{L}_{\text{p}}$, the difference of the predicted joint positions and the target joint positions, normalised over the target positions.

$$\mathcal{L}_{\text{p}} = \frac{\mathbb{E}[y^p - x^p]^2}{\sum(y^p)^2 \cdot \sum(x^p)^2)} \tag{25}$$

- **Rotational error,** $\mathcal{L}_{\text{r}}$, the mean squared rotation error of the predicted joint rotations and the target joint rotations.

$$\mathcal{L}_{\text{r}} = \text{MSE}(y^r, x^r) \tag{26}$$

- **Rotation variance,** $\mathcal{L}_{\text{rv}}$, the variance of the rotational changes along the timeline. The lower variance the smoother movements.

$$\mathcal{L}_{\text{rv}} = \mathbb{E}[(y^r - \mathbb{E}[y^r])^2] \tag{27}$$

- **Adversarial error,** $\mathcal{L}_{\text{adv}}$, the adversarial error produced by the LSGAN discriminator.

$$\mathcal{L}_{\text{adv}} = \text{D}(X^S) \tag{28}$$

- **KL-loss,** $\mathcal{L}_{\text{KL}}$, the KL-divergence for probabilistic models.

$$\mathcal{L}_{\text{KL}} = \text{KL}(X^S) \tag{29}$$

- **Contact accuracy,** $\mathcal{L}_{\text{acc}}$, the accuracy of reaching the targets (with distance < 0.1m).

$$\mathcal{L}_{\text{acc}} = \frac{1}{T_s} \sum_{i=1}^{T_s} \text{Contact}(T_i, p_j) \tag{30}$$

where Contact(·) is a function that returns 1 if the target is reached ($||T^p - p_j|| < 0.1$), T_s is the total number of targets and p_j is the position of joint j.

- **Sum of position cost,** $\mathcal{L}_{\text{sp}}$, the area under the curve (AUC) for the position cost convergence.

$$\mathcal{L}_{\text{sp}} = \sum_{i=1}^{299} C_i^p \tag{31}$$

- **Sum of rotation cost,** $\mathcal{L}_{\text{sr}}$, the AUC for the rotation cost convergence.

$$\mathcal{L}_{\text{sr}} = \sum_{i=1}^{299} C_i^r \tag{32}$$

The following properties are used for quantifying the complexity of the models:

- Number of learnable parameters.
- Number of float point operations (FLOPs) for inference.
- Memory footprints.

5 Results

6 Discussion

7 Conclusion

7.1 Future Work

Appendix - Contents