DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics: Games Engineering

Deep Learning Approach for Motion Control of a Physically Simulated Humanoid Avatar in VR

Lukas Goll

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics: Games Engineering

Deep Learning Approach for Motion Control of a Physically Simulated Humanoid Avatar in VR

Deep Learning-Ansatz zur Bewegungssteuerung eines physikalisch simulierten humanoiden Avatars in VR

Author:	Lukas Goll
Supervisor:	Klinker, Gudrun Johanna; Prof. Dr.
Advisor:	Weber, Sandro; M.Sc.
Submission Date:	15.12.2020

I confirm that this master's thesis in informatics: games engineering is my own work and I have documented all sources and material used.

Munich, 15.12.2020 Lukas Goll

Acknowledgements

I would like to thank the university and the chair for Augmented Reality represented by Prof. Gudrun Klinker to make it possible to write the master thesis, especially during these difficult times with the Corona pandemic. I would like to thank my advisor Sandro Weber, who gave great support in answering some questions and supported my efforts with telecommunication, since meetings in person were not possible. Finally, I would like to thank my family supporting me in this endeavor.

Abstract

In this work, motion control of a physically simulated humanoid avatar using a neural network is explored. With two multi segmented human shaped avatars, one animated by designed animation and the other supposed to follow, we create a scenario in which the follower is controlled by a neural network. We recorded datasets and explored through supervised learning how the neural network as adaptive controller can learn to actuate the avatar. This required extensive hyperparameter tuning. Findings of the training sessions are documented and analysed. Two neural network architectures are proposed and evaluated.

Acronyms

VR	Virtual Reality
DoF	Degrees of Freedom
HMD	Head Mounted Display
IK	Inverse Kinematics
GE	Game Engine
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
MSE	Mean Squared Error

Contents

1. Introduction

1.1. Introduction to Virtual Reality

Currently Virtual Reality (VR) has overcome the trough of disillusionment [LF03] and reached the slope of enlightenment, as Head Mounted Display (HMD) devices become more and more common in the entertainment sector. Whereas VR has been subject of research for a long time (see 2.1), the market and demand for VR has grown strong in the last couple of years, and there are predictions for growth. [Res19] [Tan20].
Even though this technology remains expensive and uncomfortable to use, HMDs are heavy and the technical requirements for high frequency rendering are expensive. But its multi-purpose potential in research and exciting prospects in entertainment give rise to many multimedia companies investing [Mar+17], however the demand for high-end VR devices has been extraordinary, which can be seen from the availability of VR devices. With the release of Half Life Alyx, Valve's highly anticipated VR video game, orders of the Valve Index, a HMD device with top-shelf specs and finger tracking capable controllers, had been delayed to 8-10 weeks [Wil20], due to its high demand. These sales suggest that there is a huge interest in these technologies, in spite of of their price and ergonomics
In VR applications, users are able to see and interact with virtual objects. Often, the user is represented by his avatar. Through this, the user has an orientation and position in the virtual space and sometimes even a body. In section 2.1, we consider VR from a theoretical viewpoint and identify presence, the feeling of being there [Sla09], as the core factor for success in humans benefiting from VR applications, for instance, in medical treatments [RM12]. We explain presence in section 2.1.1 in more detail. One strong implementation of presence is a virtual body [SU93] which has high fidelity, is uninterrupted and creates a strong illusion of realism. Sheridan [She92] suggests that presence can be achieved if an avatar can physically interact with its virtual environment. He describes telepresence as being physically present with virtual objects. In a Game Engine (GE) like Unity, it means that the avatar must be designed so that it is motion controlled correctly, based on the conditions imposed by the physical simulation in this GE.

1.1.1. Current state of embodiment in virtual reality

In regard to the requirements of a convincing virtual representation, current full-body tracking solutions are able to mimic the user's body in a limited way. But they fail to convince when it comes to collisions between avatar and virtual objects and especially between avatar and avatar. In many recent applications, the avatar is only represented visually, collisions between avatar and virtual objects are ignored, and the avatar moves through. Of course, this breaks the illusion of realism. It limits the degree of interactivity and therefore presence. There is a need for robust solutions to fully represent avatars physical.
In the industry, the common approach for full body representation in VR is six point tracking and pose estimation by Inverse Kinematics (IK). IK solves a human shaped pose based on the position of limited tracking points and a set of constraints [Gro+04]. The number of VR games, which support such full body tracking are rather small. The most dominant supporter for full body is VRChat [Viv20a]. VRChat is a virtual sandbox, which experiences a lot of media attention, because of its creative player base [Sta20] . It allows its users to create virtual worlds and custom avatars. These worlds can be uploaded and explored together. Since it allows multiple users to interact with each other, it creates a huge sense of social credibility and realism. It provides a huge platform for creativity and its popularity even attracted commerce [Dam19]. With full body trackers from HTC Vive [Viv20b], VRChat allows, based on the number of active trackers, four point and six point tracking. The four tracked points are the HMD, two of the HMD's controllers and one tracker, which should be located at the hip. In this configuration, the VR avatar legs are animated by animations. Adding two more trackers on each of the users feet, enables full body posing based on IK. Using the trackers is fully optional. From these tracked points, the avatar pose is inferred by IK. VRChat experienced a massive spike in its active players early 2018 [Ste20b]. Remarkably, compared to other VR Games, in particular Steam's flagship game Half Life Alyx [Ste20a], VRChat can keep an active player base [Ste20b]. This makes VRChat state of the art regarding full body tracking in the industry.

1.2. Previous Work

The Technical University of Munich currently develops a framework for physical motion control of a humanoid avatar in Unity [WK19]. The goal of this framework is to provide an avatar, which can interact physically with objects in 3D and is moved by forces and torques. Whereas an earlier solution of the framework solved the IK problem for a physical avatar to be moved based on user tracking on a external server, Karas [Kar20] proposed a control structure in Unity using a combination of Unity's Rigidbodies (3.1.2) and its joint systems. These joints require detailed configuration and tuning. For each of these joints, the Degrees of Freedom (DoF) and their angle ranges are limited to fit human physicality. Therefore, they are manually adjusted to create realistic and physically correct motions. Using these configurable joints results in disadvantages. Karas reported that configurable joints break on rapid acceleration. More than that, this setup breaks on collisions with too heavy or immovable objects. In total, this setup is too error-prone for complex scenarios such as virtual environments might be. Therefore, it has been suggested to use a neural network to actuate the humanoid avatar.

1.3. Scope

In this work, we explored an approach to train a neural network to learn how to control the motion of a human shaped avatar in a physically correct way. For this, we use Unity as framework, since it provides a solid, real time engine with integrated physics simulation. This environment and its physics simulation, PhysX, imposes rules which we consider in our approach. Based on these rules, we formulate how the humanoid avatar has to be built, then we understand how this avatar has to be moved. We design a data model, of which we expect to supervise our network. We build different neural network architectures and experiment on hyperparameters. Then the approach is evaluated followed by a conclusion. While using Unity as real time simulation, we also incorporate Ubi Interact [San20], a framework which is developed by the technical university of Munich for connecting multiple devices and environments for research and development.

2. Theory

2.1. VR

The concept of VR, has been introduced in an early study to display three dimensional objects with a HMD [Sut68] by Ivan Sutherland. It allowed users to see a simplistic but fully virtual world comprised of a cubic grid. Stereoscopic view enabled the user to see the object in 3D and head tracking made it possible to see it from different perspectives. This novel idea let researchers [Fis+87] [Fol87] anticipate VR as a more sophisticated way of human machine interaction and machine accessibility. Fisher [Fis+87] et. al. virtual environment display system even incorporated a data glove, with the purpose that users are able to interact with virtual objects. Since then, VR and its research progressed from theoretical concepts to actual implementation of devices, software and guidelines. The virtual reality triangle, proposed by [BC03], defines with I^3 the three most important key components for a successful VR application. The three I's stand for Immersion, Interaction and Imagination. It is immersive, because senses of the real world are excluded. It is interactive, so that users are actively engaging with virtual objects. And it's imaginative as users are able to produce mental sensations or emotions which are the same as real ones. If a virtual reality application is successful in archiving these three properties, researchers have explored multiple applications, which are beneficial for humans in many ways. For instance, there is strong evidence that VR can be used to treat phobias [Rot+00] [Bot+00] and mental conditions such as PTSD [Rot+99]. There's also research suggesting that VR can be used as mediator for better teaching [Kan+13].

2.1.1. Presence

Studies have explored, under which conditions a user is affected and engaged in VR. While the previous I^3 describe objectively a good VR application, researchers use the term presence as a more subjective feeling of being there, inside VR. Slater [Sla09] describes presence as a place illusion. The feeling of being there is described by him as believing the virtual place as a real place, which is visited. He also introduces the plausibility illusion, which "refers to the illusion that the scenario being depicted is actually occurring" [Sla09]. He proposes that, if virtual applications are able to create both illusions to the user, he responds to the virtual world as a real world. His observations are covered by other studies [Die+15] [PA07], which come to the conclusion that presence becomes the mediator for emotional investment in virtual scenarios. Users need to arrive in the virtual to be affected by VR treatments. To have a virtual avatar is stated to be very important for presence [She92] [SSC10].

2.2. Humanoid Avatar

In research, there exists attempts to create physic based locomotion for virtual humanoid like bodies. Mostly, the human body has been constructed differently. [AF09] Allen and Faloutsos modelled the body downwards from the spine, with two hip joints, knee joints and ankle joints. The focus of their work was to learn bipedal locomotion by creating torques for these joints. Reil and Husbands [RH02] created a body by separating the joints as representations of DoF, from the physical bodies which represent mass in the simulation. They use a rigidbody simulation and actuate the body by forces and torques. These approaches differ from ours, since they have a heavy focus on learning bipedal walking for robots. In contrast, we want to create a full body representation of a human in VR. Our avatar should include the arms, spine, chest and the head. Mittman et al. [MFS07] model is more complete. Their model refers to the hierarchical skeleton model, in which segments are connected with bones and joints. This hierarchy is used in 3d animation and modelling software and Unity [Tec20a], although with varying complexity. Mittman et al. describe their humanoid model in detail. It's joints are limited to humanoid fitting DoF and limits. They move the avatar with physic based animation, by constructing a detailed physics based body model, with calculated masses for each segment. This approach imposes strict definitions on the body, as the ranges of joint extends are predefined. [SBR12] et al. used a hierarchical bone structure which is only moved on its joints with rotations. They used motion capture to collect 10 minutes of data which trained a neural network. To avoid absolute rotations and positions they worked with relative quaternion only.
Our neural network approach tries to impose as few regulations and manual tuning as possible. Since we want to explore what the network is able to learn on its own. While some of the earlier described models only use rotations, we want to enable our network to work with translation, too. Our avatar is modelled around the hierarchical skeleton model, too. The hierarchy structure can be observed in Figure 2.1

The hip bone is the root of the hierarchy, from there the tree like structure reaches into the hands, feet and head. Whereas other approaches model ankles and other joints with DoF, our model joints can rotate freely. Since the model needs to be represented in Unities physical simulation, we model physical surfaces around that hierarchy. More

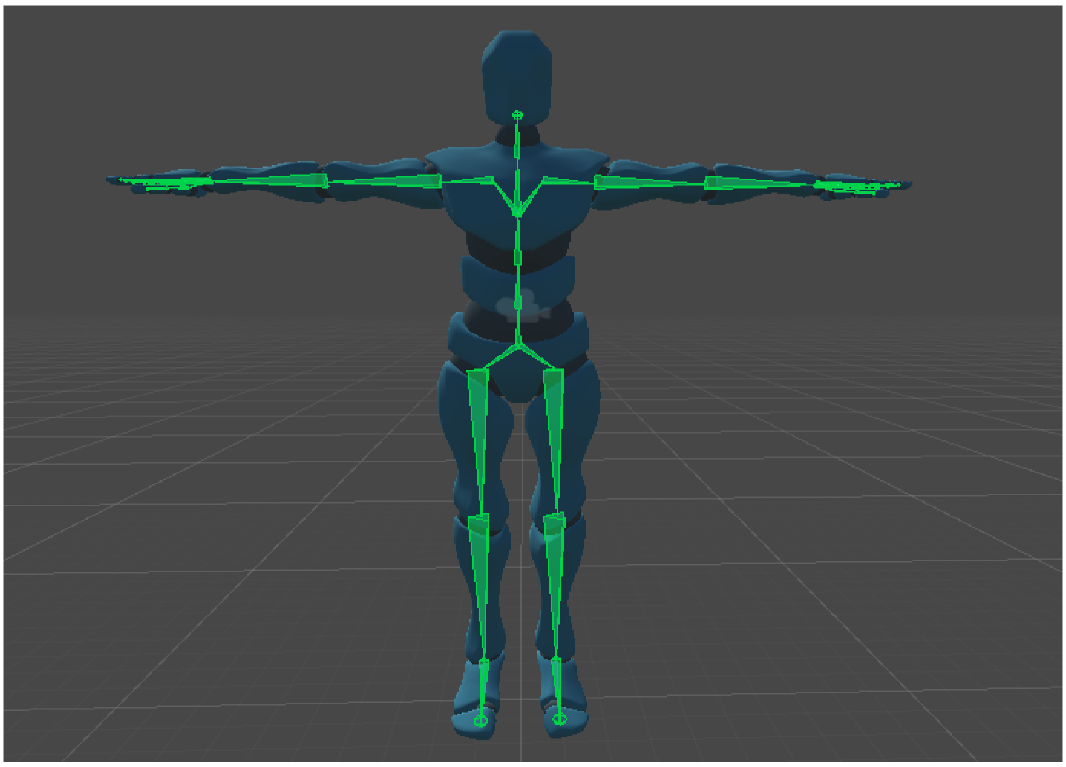


Figure 2.1.: The Hierarchical skeleton model begins from the central hip to its extremities. The hipbone is visible as the three directional bone in the hip area of the avatar.

on that in section 3.2.

2.3. Motion Control

Motion control refers to the idea of operating virtual [Cav+98] or non virtual [Pad+16] objects to perform motion to reach destinations or to fulfil tasks. Especially in the field of robotics, motion control is researched in particular. Operating robots from afar shows great potential in preventing people from putting their lives at risk in harmful environments. This motion control of robots draws similarities to the motion control of a virtual body that is physically simulated in VR. In both cases, there is an optimal motion that has to be performed. Velagic et al [VOL08] use a recurrent neural network to learn linear velocities given positional errors of the robot. They show that it is possible to infer velocities and to use a neural network to control the motion of a robot. Aparanji et al [AWA17] use a neural network to control the rotational motion of joints of a multi segmented robot. These and other approaches [Ooe+13] [Miy+88] [Ara10] [Kaw90] identify the problem of adaptive motion control as a cyclic system:

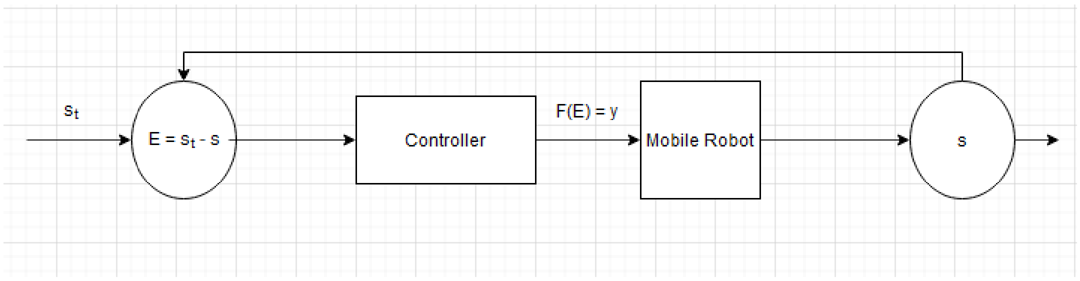


Figure 2.2.: State s, desired state s_t, actuators y, error state E.

The robot exists in 3D space with its multi segmented body. Therefore, its current state s can be described in multiple ways. For instance, with positions and rotations of the robot's segments. Since the robot has to follow a desired movement s_t, which differs to the current state of the robot, there exists an error between those two states. An adaptive motion control system corrects these errors by creating the required actuators for all segments, so that the robot moves properly to minimize these errors.

2.4. Neural Network

Compared to fully analytical solutions, like pd or pid controllers [Kar20][Ara10], using a neural network yields potential advantages. From previous work, we know that creating physical movement through intensive setup of configurable joints in Unity is error-prone, therefore we explore how a neural network can infer humanoid movement based on physical actuation and rigidbodies in Unity. With different architectures and ideas, we try to find a network which imposes motion control. A neural network can find solutions to complex problems, solutions a human can not identify. We want to answer the question of whenever a network can impose correct rules on how to actuate an rigidbody avatar based on its current state.

2.4.1. Neural Network Introduction

A neural network is a feed forward network which consists of multiple layers. It starts with input nodes, basically a node for each input value, and ends with output nodes, for which the number of nodes is dependent on the problem. On classification problems, the number of output nodes are the same as the number of distinct classes [KH10]. On regression problems, the number of output nodes is equal to the number of output values. In between each input and output layer are one or many hidden layer, which have the following structure [RN16a] [Goo+16a]:

$$o_y = g(\sum_{x=1}^{n} w_{i,y} * x_i + b_y)$$

o_y is the output of the y-th node in the network, with the activation function $g(.)$, weight $w_{i,y}$ that connects this note with the i-th note of the previous layer. x_i is the i-th input of this node and b_y its bias.

2.4.2. Optimization

A neural network learns through optimization. Given a loss function ΔE, which depicts the error of a networks prediction, the network optimizes with $\frac{\delta \Delta E}{\delta \theta} = \Delta$. θ is any trainable parameter of the network, such as its weights and biases. With backpropagation, Δ for each θ can be calculated. Then it is possible to optimize the network with gradient descent [RN16b],

$$\forall \; \theta \; \epsilon \; \Theta : \theta \leftarrow \theta - \alpha \Delta$$

with Θ as set of all trainable parameters of the network and α as real scalar as learning rate. Note that this is an iterative approach, because the optimal learning rate and direction to solve for $\Delta E = 0$ is unknown. More elaborate optimization techniques have been developed, such as Adagrad[DHS11]. Adagrad self regulates its learning rate α. It removes the need to manually tune the learning rate, because it regularizes features based on the optimization topology [Sta16]. The current erroneous state directly affects

α since it is multiplied by $B = diag(\sum \Delta(\Delta)^T)$. Therefore Δ directly affects the learning rate. Adam [KB14] is an extension to the Adagrad optimization. It uses bias correction to improve optimization once optimization gradients become smaller. Therefore it is a common choice as optimization technique.

2.4.3. Neural Network Function Theory

Let's assume there is a function f(.) that moves an humanoid avatar physically correct. With the universal approximation theorem it can be proven that a neural network with one hidden layer using n neurons can approximate uniformly well a continuous function defined on the interval of $[0, 1]$ [Csá+01].

Let g(.) be an arbitrary activation function and C as set of all continuous functions, then $\forall f \epsilon C([0, 1]), \forall \epsilon > 0 : \exists n \epsilon N, w_i, a_i, b_i, i \epsilon 0...n$:

$$A_n f(x) = \sum_{j=1}^{n} (v_j * g(\sum_{i=1}^{m} w_{i,j} * x_i + b_j))$$

let $A_n f(x)$ be an approximation of the function f(.) with m inputs, then

$$sup_{x \epsilon [0,1]} |(A_n f(x) - f(x)| < \epsilon$$

The theorem entails that there exists a network with n nodes which can approximate f well enough [JKK20], but as Goodfellow [Goo+16b] et al. and Du et al. [Du+19] point out, with increasing deeper or larger networks, they become harder to train. This causes two problems. First bloating up the only hidden layer in a neural network causes the network to grow in trained parameters drastically. While deeper networks suffer under the vanishing and exploding gradient problem [Goo+16b]. Shallow networks are subjects of discussion compared to deep networks [MP16]. Under the right conditions, deeper networks can reduce the complexity of approximation and training.
Even though the universal approximation theorem states that each continuous function can be approximated, in practice it requires the network to have the required architecture and data. To find an appropriate approximation of the wanted function requires a certain shape of hidden layer. All these choices and variants are observed with a set of hyperparameters which requires tuning.

2.4.4. Overfitting

Another consideration that has to be taken into account is overfitting. The network learns to approximate a function f, based on the data samples it observes during training. At some point, the network reduces the approximation error, but fails to

improve on data which is not part of the training data. Often the performance on unseen data even degenerates, although we continue to train the network. This problem is called overfitting [Haw04]. When it comes to regression problems such as finding the function f is, it has to be considered.
Therefore, it is common practice to split the available data into two chunks. Whereas any machine learning algorithm trains with the training data, its ability to generalize on unseen data is observed by testing the algorithm with data points from the test set.

Prevention

Three common approaches are used to prevent overfitting: regularization penalties, dropout and early stopping.

First, we discuss regularization. With L1 regularization, the network's loss is penalized by the absolute sum of the weights [Goo+16c].

$$L1_{reg} = Loss + \lambda * \sum |w|$$

Whereas the sum of squared weights is referred to as L2 regularization [Goo+16d].

$$L2_{reg} = Loss + \lambda * \sum ||w||^2$$

Usually, L2 is preferred over L1, because L2 penalizes weights of the network by reducing them, without forcing them to zero like L1 and killing the activation. Notice the lambda λ in the equations above. It is a real value between zero and one. The best choice for this lambda is problem dependent, since the function to be approximated by the network depends on the problem to solve. Regularizing the loss function is a very common approach in Regression problems, as it helps to solve ill posed problems [PTK87]. Let's imagine, we try to find a function f that maps from our input X to our desired values y. As we record our data we are limited to the samples of our survey. Girosi [GJP95] et al. describe the problem as ill-posed, since our input might be subject to noise, and the number of solutions are infinite. Therefore, we need to add a prior, for instance assuming that our solution is smooth. This can be achieved by adding regularization penalty to our loss during training time.

Another common technique is dropout. It is applied on network layers with a real value between zero and one. Let's assume dropout with a value of 0.5 has been chosen. During training, weak nodes of this layer are pruned, so that half of the network remains. It has been proven that dropout can improve the network to generalize better [Sri+14]. Zaremba et al. [ZSV14] show that dropout can even be used on LSTM models

2.4.6, if applied properly.

Another technique to prevent overfitting is batch normalization. For many activation functions, for instance sigmoid activation, the network faces a problem which is referred to as vanishing gradient problem [Hoc98]. This appears when the nodes input continue to grow but the activation function shape maintains the same output. To tackle this issue, the output of the previous layer can be normalized. According to Ioffe and Szegedy, batch normalization can improve generalization and network learning speed [IS15].

During training, sooner or later the network reaches a state, where the network infers knowledge which doesn't generalize. Plotting the training and test loss reveals it, as the training curve continues to descend, but test loss converges to a value above training loss. Sometimes, test loss even worsens as it diverges upwards.

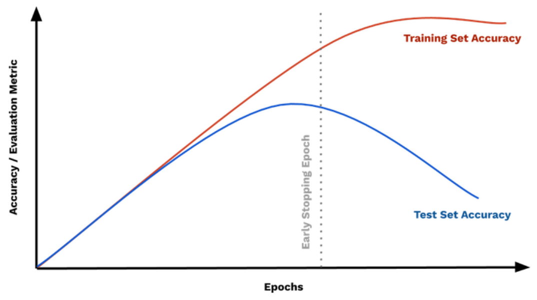


Figure 2.3.: Graph showing training and validation loss in an overfit scenario [DLA20].

Because of this, it is common to stop training early to prevent the network to degenerate [Goo+16e].

2.4.5. Hyperparameter Tuning

Previously, we discussed the performance of a network is highly dependent on the choices of variables as we construct a network. This set of variables is called hyperparameters. Finding the optimal set of variables requires a trial and error process [SD91]. Most often, the variables are found with grid search or random search strategies [NK]. Since it is impossible to try all hyperparameter combinations and values, it requires a mix of expertise and pragmatism. In this work, we explore different configurations of hyperparameters to find a good solution and to find the function that maps the avatar error states to actuations. This process of finding the best set of hyperparameters is called hyperparameter tuning. To prevent any bias during this process, the test data is split again in two sets. The first set, remains the test data for our training. Based on the performance of the test data during training sessions we choose hyperparameter and early stopping. Whereas the second set of data, the performance dataset, is a unbiased indicator for performance regardless of the training process.

2.4.6. Recurrent Neural Network

To move a humanoid avatar physically, the network needs to observe the current velocity of the avatar's segments in some way. In computer graphics and physics simulations, time is processed in discrete time steps. Let's observe the two pictures below.

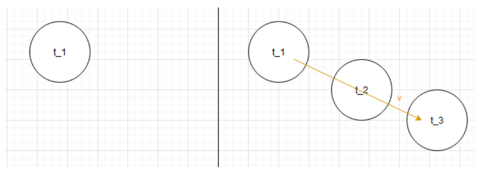


Figure 2.4.: On the left side only one observation is not enough to estimate the circle's velocity. On the right, more observation allow to estimate the circle's velocity.

When we observe a physical body in space once, we can't estimate its motion. There is not enough information to make any statement, whenever the ball is going to move in the next timestep. Whereas more observations from multiple time steps allow us to predict the body's velocity to some extent. The velocity of a body in 3d space is a sequence of previous positions.
For sequential data, such as a series of observations, recurrent network layers are build to process information on an temporal dimension.

A recurrent layer with an arbitrary sequence length can be rolled out in the temporal domain, processing a sequence of data values along the temporal axis [AS20] (see 2.5). While recurrent layers have output just as regular networks, recurrent cells feed internal information to themselves. The internal state is passed through along the temporal axis. Therefore, the recurrent layer can accumulate information over time.

Long Short Term Memory

A simple, recurrent structure, as we discussed above, faces a serious issue: the vanishing gradient problem [Hoc98]. While a recurrent network layer is rolled out along the temporal domain, it favours short term memory.

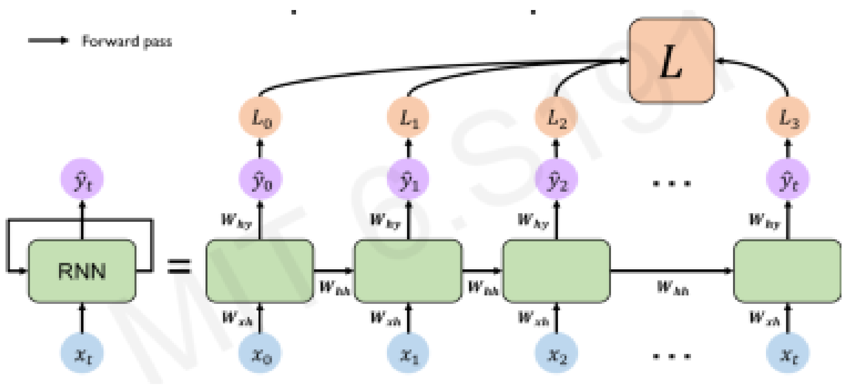


Figure 2.5.: A recurrent network is rolled out along its temporal axis. [AS20]

A standard recurrent layer will be biased to learn information at the end of sequence, hence it has short term memory. This occurs because during training, the error propagates backwards through the sequence. The gradient vanishes with each connection exponentially. One small gradient in the previous step of the sequence will degenerate further, hence the network will learn less from older sequence steps. To counter this issue, LSTM models have been designed [HS97].

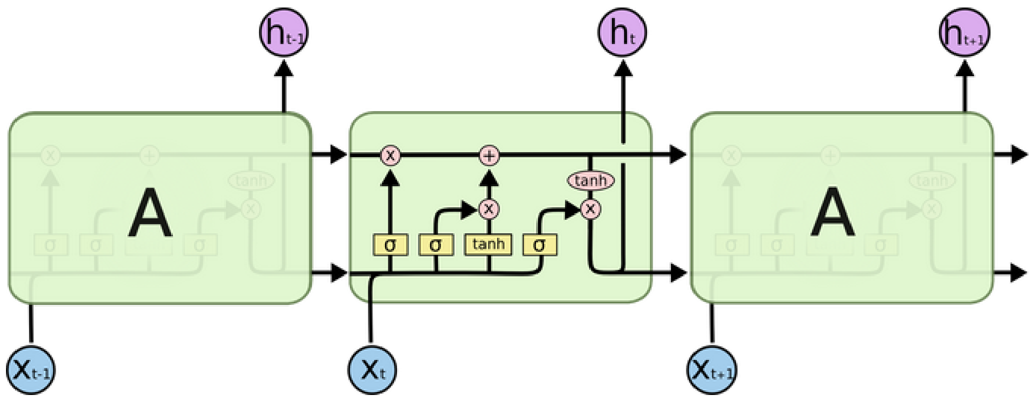


Figure 2.6.: LSTM module visualized [AS20].

They consist of multiple gates, which regulate the amount of information which should stay in memory, hence what should be forgotten. In figure 2.6 you can see the contents of the LSTM cell. Notice how the internal information is recurrent along the upper horizontal line. This line represents the memory of the LSTM layer. The lower horizontal line is the input line of the LSTM cell. It takes in the output of previous timestep $t-1$, and the input value of the sequence at timestep t. The First gate is the forget gate. It's the first sigmoid function in the LSTM module. The sigmoid output ranges from zero to one. As you can see along the memory line, its contents are multiplied with the result of the forget gate. If the forget gate produces a zero, the cell forgets everything previously memorized. Note, this only affects the cell's recurrent flow, not the output. Next, the memory is updated. We process our new cell state with a tanh activation from the sequence input and the previous cell output. And we see again a sigmoid gate, which forgets parts from our new cell state. This new cell update is added to the LSTM memory by addition. In the last step, the output of the cell is calculated by applying the tanh function on our memory. With the last sigmoid gate, we decide how much of our output leaves the cell. Because of the vanishing gradient problem of sigmoid and tanh functions operating in the RNN network, the optimal range of inputs needs to be considered [Hoc98]. The usage of LSTM cells in a recurrent neural network layer, allows networks to maintain long or short term memory. It will learn on its own, what parts of the sequence are important.

2.4.7. Data Processing

Neural networks operate on certain ranges of values better, because of their activation functions. This is even more relevant for recurrent neural networks because they use sigmoid and tanh activations. Goodfellow [Goo+16f] et al. state that staying in a favourable parameter space is a good chance of avoiding the vanishing and exploding gradient problem. Therefore, it is common to pass the data preprocessed.

MinMax Scaling

The first common preprocessing technique is called MinMax Scaling. It is the go-to method on data, which is not normal distributed [Bro20a]. In this technique, the data is scaled along its features linear to fit them in a specified interval. Which means each feature has a maximum value *max* and a minimum value *min*. With these values MinMax scaling is applied on values x of that feature to scale it linear into the interval [a,b] [dev20a].

$$\hat{x}_{minmax} = \frac{(x - min) * (b - a)}{max - min} + a$$

This operation is fully reversible by solving for $\hat{x}_{minmax}$:

$$\begin{aligned}
x &= \frac{(\hat{x}_{minmax} - a) * (max - min)}{b - a} + min \\
(x - min) &= \frac{(\hat{x}_{minmax} - a) * (max - min)}{b - a} \\
\frac{(x - min) * (b - a)}{max - min} &= \hat{x}_{minmax} - a \\
\frac{(x - min) * (b - a)}{max - min} &+ a = \hat{x}_{minmax}
\end{aligned}$$

Standard Scaling

Is the data standard distributed, Brownlee [Bro20a] suggests to preprocesses the data with standard scaling. We also see standard scaling in the works of Zhang et al. [Zha+18] and Grzeszczuk et al. [GTH98]. With the feature's mean μ and standard derivation σ, it removes that mean and scales the feature to unit variance [dev20b].

$$\hat{x}_{standard} = \frac{x - \mu}{\sigma}$$

This operation is fully reversible by solving for $x_{standard}$

$$\begin{aligned}
x &= \hat{x}_{standard} * \sigma + \mu \\
x - \mu &= \hat{x}_{standard} * \sigma \\
\frac{x - \mu}{\sigma} &= \hat{x}_{standard}
\end{aligned}$$

3. Methodology

In this chapter, we formulate a neural network based approach to motion control a humanoid avatar physically simulated. The following sections discuss the planning and the details of our approach in regard to the following: the environment the avatar is simulated in and how to move the avatar physically correctly. The creation of the dataset, which is used to train a network to become the avatar's controller. Strategies and architectures of our hyperparameter training and the Ubi Network, which we use for communication between Unity and a trained neural network model.

3.1. Unity as Environment

We chose Unity as environment for our experimentation. This GE comes with PhysX [Tec], a Rigidbody guided simulation in typical frame-wise computation. As usual for Game Engines, a running instance cycles a game loop at runtime, which processes arranged events necessary for a running game. For the physical simulation, it is important to mention that physics computation is done in the internal physics update at a fixed time rate. For every 0.02 seconds, a physics update is processed. In the FixedUpdate function, we can parametrize physics components before they are simulated.

Unity as a simulation environment imposes two important aspects for the correct implementation of physical correct movement. Physics calculation and interaction happens only in the internal physics update. Second, it is required to input actuators before the internal physics update.

3.1.1. Scene and GameObjects

In Unity, a scene is an editable setting of multiple virtual objects in a 3-dimensional coordinate system. All of these objects are either relative to the origin of this space, or are related as child objects to other parent objects. This gives these objects a world transform, which comprises of position, rotation and scale. The world transform differs to the relative transform, because its values are unrelated to any parenting. Objects in Unity are referred to as GameObjects. They allows Components and Behaviours to be attached. Behaviours allow execution of code, while Components provide attributes.

3.1.2. Rigidbody

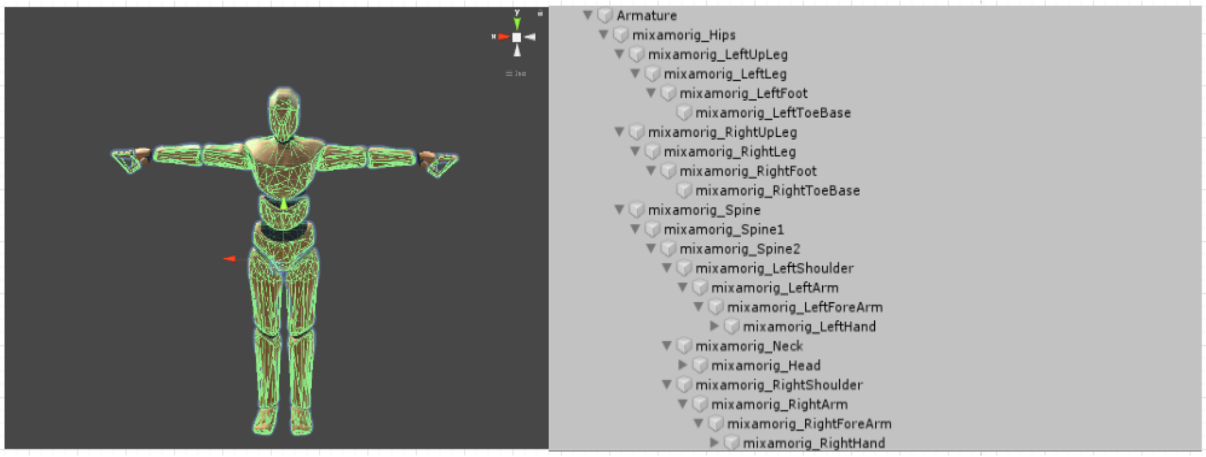
Objects that are to be considered as physical objects require that they have a rigidbody component attached. Let's name GameObjects, which have a rigidbody component attached simply Rigidbodies. They are set to dynamic, which means they are able to collide with each other and can move by physical forces. They don't respond to a gravitational pull along the y-Axis since we deactivate it on the component.

The Unity API allows two ways to move Rigidbodies. One way is to move them directly to a target destination or rotation [Tec20d] [Tec20e]. The other method imposes forces on the Rigidbody with AddForce and AddTorque [Tec20b] [Tec20c]. As it is specified by PhysX guidelines [Cor], the best way to move objects physically is by applying force. To produce more correct physical actuation, it is advisable to use force and torque, because "[t]he advantage of forces is that regardless of what forces you apply to the bodies in the scene, the simulation will be able to keep all the defined constraints (joints and contacts) satisfied."[Cor]. Therefore in Unity, Rigidbodies actuate most physical-friendly with force and torque. If we want to motion control a humanoid avatar, we need to find a function that applies forces and torques to the Rigidbodies that comprise the avatar. Let's see how an avatar, consisting of Rigidbodies, can be constructed.

3.2. Rigidbody Avatar

The Rigidbody avatar should be a body that represents a human in the virtual environment. From the previous chapter 2.1.1, we know that the humanoid avatar should represent a human shape and ideally be able to interact with the environment as much as possible to create a huge sense of presence. To create an avatar in Unity which interacts physically with Rigidbodies, we need a avatar with human shape consisting of some Rigidbodies. In this project, the avatar has been designed around the humanoid hierarchical bone structure. This structure is used for animating human characters and motions in video games and 3d animation. The humanoid rig standardizes a hierarchy of bones which move a human like 3d surface. These humanoid animations directly move rig bones. Points of the avatar surface are moved related to their closest bones. This allows different avatar designs to follow the same motion imposed by an animation, independent from the avatar's topology. This allows us to construct a humanoid avatar, which has not only a humanoid bone structure, but also Rigidbodies.

Notice on the figure 3.1, that this setup allows an arbitrary amount of surfaces. While the hierarchy includes three spine bones, the manikin contains one lower surface above the hip and one upper surface as chest piece.



```
▼ Armature
  ▼ mixamorig_Hips
    ▼ mixamorig_LeftUpLeg
      ▼ mixamorig_LeftLeg
        ▼ mixamorig_LeftFoot
            mixamorig_LeftToeBase
    ▼ mixamorig_RightUpLeg
      ▼ mixamorig_RightLeg
        ▼ mixamorig_RightFoot
            mixamorig_RightToeBase
    ▼ mixamorig_Spine
      ▼ mixamorig_Spine1
        ▼ mixamorig_Spine2
          ▼ mixamorig_LeftShoulder
            ▼ mixamorig_LeftArm
              ▼ mixamorig_LeftForeArm
                ▶ mixamorig_LeftHand
          ▼ mixamorig_Neck
            ▶ mixamorig_Head
          ▼ mixamorig_RightShoulder
            ▼ mixamorig_RightArm
              ▼ mixamorig_RightForeArm
                ▶ mixamorig_RightHand
```

Figure 3.1.: On the left: The 3d model of a humanoid shaped avatar. The green areas are the surfaces that can collide. On the right: The hierarchical bone structure. All of these bones make a GameObject in Unity, with it's own position and rotation.

3.2.1. The 3D Model

For all of our experiments we used this manikin model, which is freely available for download on Adobe's Webside Mixamo [Ado]. As visible in figure 3.1, the avatar consists of dark spheres, which represent visual joints, and more complex shapes, which are the avatar's surfaces. These surfaces contain mesh colliders, which define the surface shape and its volumes for collisions. These volumes can be seen in the figure 3.1 as green outline. More details on the implementation of this Rigidbody avatar in section 4.2.3. So we construct a 3d model, which can be animated by humanoid animations and comprises of surfaces as Rigidbodies.

3.2.2. Training from Synthetic Data

Using an avatar with a hierarchical bone structure, allows us to move our avatar with minimal effort. We can utilize animations from 3d entertainment. Not only that, we can simply rely on animations from different sources like Mixamo [Ado]. Every animation that animates the hierarchical bone structure could be used. Since those motions are directly mapped to the armature of the human rig, animations from different sources apply on the same rig and allow us to accumulate much more data. Using synthetic datasets have the advantage to have data much easier available and have been topic of research recently [Jad+14] [Xu+19].

3.3. Motion Control of a Physical Avatar

Since we are familiar with the Unity Environment and our virtual avatar as a representation of a human body in virtual space we can formulate an approach to solve our problem. How is it possible to control the motion of a virtual avatar in Unity, in a physically correct way ?

Let's define a state, s_t as the state of the humanoid avatar in simulation at time t. Time between states s_t, s_{t+1}, s_{t+2}... shall be constant and depicted by Δt. In Unity, Δt is set to a fixed rate of 0.02 seconds. On each state s_t , the bones have a world position $p_t \epsilon \mathbb{R}^3$ and world rotation $r_t \epsilon \mathbb{R}^3$ in 3D space at time step t in our simulation.

3.3.1. Motion

Now we create a sequence of states which set the motion for our avatar. Let's assume that for each s_t a desired state ts_t or target state, which establishes a target position $tp_t \epsilon \mathbb{R}^3$ and target rotation $tr_t \epsilon \mathbb{R}^3$ for each bone. For each time step t the correctness of our motion is defined as:

$$CM(s_t) < - > s_t = ts_{t-1} < - > p_t = tp_{t-1} \quad \text{and} \quad r_t = tr_{t-1}$$

.

Our avatar begins at $t = 0$ in a resting pose, called t-pose, which is commonly used in 3d media. From there, we control our avatar by determining for each $t > 0$ a desired target state. With $T \epsilon \mathbb{N}$ as set of all time steps simulating, our avatar moves correctly if

$$\forall t \epsilon T : CM(s_t) = 0$$

To solve $CM(s_{t+1})$ at t, we introduce

$$e_p(t) \epsilon \mathbb{R}^3 = tp_t - p_t$$

as positional error at timestep t and

$$e_r(t) \epsilon \mathbb{R}^3 = tr_t - r_t$$

as rotational error at timestep t.
To maintain the motion correctly we need to solve for $CM(s_{t+1})$

$$CM(s_{t+1}) < - > s_{t+1} = ts_t < - > p_{t+1} = tp_t \quad \text{and} \quad r_{t+1} = tr_t$$

By translating and rotating bones of the armature on every time step by the positional and rotational error yields the correct motion $CM(s_{t+1})$:

$$p_{t+1} = p_t + e_p(t) \quad r_{t+1} = r_t + e_r(t)$$

$$p_{t+1} = p_t + (tp_t - p_t) \quad r_{t+1} = r_t + (tr_t - r_t)$$

$$p_{t+1} = tp_t \quad r_{t+1} = tr_t$$

Further assuming a resting pose where $p_0 = 0$ and $t_0 = 0$:

$$p_1 = p_0 + e_p(0) \quad r_1 = r_0 + e_r(0)$$

$$p_1 = 0 + (tp_0 - 0) \quad r_1 = 0 + (tr_0 - 0)$$

$$p_1 = tp_0 \quad r_1 = tr_0$$

Therefore, a trivial solution is found. But as we know, Unity's PhysX environment requires Rigidbodies to be moved by forces and torques.

3.3.2. Force

The Force $F_0(t)\epsilon\mathbb{R}^3$, which is required to move a Rigidbody in Unity from p_t to tp_t in a given window of time Δt is the distance $\Delta p = e_p(t)$ multiplied by its mass m_t derived by Δt twice [Bou02a],

$$F_0(t) = m_t * e_p(t)/(\Delta t)^2$$

assuming the Rigidbody is not subject to any velocity.

Let's assume the Rigidbody was subject to forces in our simulation already, then we depict its velocity at timestep t with $V(t)$

If we consider $V(t)$ acting on its Rigidbody, the force $F(t)$ to move it to a desired position tp_t is

$$F(t) = m_t * (V_{new}(t) - V(t))/(\Delta t)$$

with $V_{new}(t)$ as the velocity, which is the velocity given by $V_{new}(t) = e_p(t)/(\Delta t)$

$F(t)$ is the actuation in force, which is necessary to translate the body physically correctly at timestep t.

3.3.3. Torque

The Torque $T_0(t)\epsilon\mathbb{R}^3$ which is required to rotate a Rigidbody in Unity from r_t to tr_t in a given window of time Δt is the distance $\Delta r = e_r(t)$ multiplied by its Inertia Tensor $I(t)$ derived by Δt twice [Bou02b]

$$T_0(t) = I(t) * e_r(t)/(\Delta t)^2$$

assuming the Rigidbody is not subject to any angular velocity.

Let's assume the Rigidbody was subject to torque in our simulation already, then we depict its angular velocity at timestep t as $\omega(t)$

If we consider $\omega(t)$ acting on its Rigidbody, the Torque $T(t)$ to rotate it to a desired rotation tr_t is

$$T(t) = I(t) * (\omega_{new}(t) - \omega(t))/(\Delta t)$$

with $\omega_{new}(t)$ as the angular velocity, which is given by $\omega_{new}(t) = e_r(t)/(\Delta t)$

$T(t)$ is the actuation in torque, which is necessary to translate the body physically correctly at timestep t.

3.3.4. The Dataset Model

We define input X and output y. Given X and y in the form of the dataset, the network will try to learn a function $f : X| - > y$.

The input X for our neural network are the positional and rotational errors $e_p(t)$ and $e_r(t)$ for each bone at timestep t. Therefore, X is a matrix with the shape $(m, b * 2 * \mathbb{R}^3) = (m, b * 6)$ with m as number of consecutive time steps recorded and b the number of bones.

Using the positional and rotational error as X as representation of the current state can be seen in the works of Velagic et al. [VOL08], Grzeszczuk [GTH98] et al. and Miyamoto et al. [Miy+88]. It has the advantage that these values are relative and therefore independent to the absolute position and rotation of the avatar in the simulation. This is an advantage for the network because this allows the network to make use of the data regardless of where the avatar stands in the simulation [SBR12]. Another advantage of using relative values as X is that in between frequent consecutive time steps, we can expect only small changes resulting in small values which should improve network performance.

For output y of our dataset we don't use the actuators $F(t)$ and $T(t)$. Rather we record for each time step t and each bone b, $V_{new}(t)$ and $\omega_{new}(t)$. We do this, because of the following reasons. First of all, during simulation $V_{new}(t)$ and $\omega_{new}(t)$ are the only values we need to calculate $F(t)$ and $T(t)$. $V(t)$ and $\omega(t)$ are given at runtime. Using $V_{new}(t)$ and $\omega_{new}(t)$ makes the mapping from X to y definitely, because it is independent of the Rigidbodies velocity and angular velocity. Because of this, $V_{new}(t)$ and $\omega_{new}(t)$ can be a substitute to calculate $F(t)$ and $T(t)$ (see section 3.3.2 and 3.3.3). Second, these values do not take the mass or the inertia of the Rigidbody into account. They are less huge, since $\Delta t < 1$, the second derivation in time increases the value, which is less optimal for a network, which operates best with lower values (see section 2.4.6). y shall consist of the new velocities and angular velocities, which form with the Rigidbodies current velocity and angular velocity the force and torque we need. Therefore, y has the shape $(m, b * 2 * \mathbb{R}^3) = (m, b * 6)$, too.

3.3.5. Dataset Generation

As we discussed previously, we want to explore if training from a synthetic data is possible. From the webside mixamo, we downloaded multiple humanoid animations. The motions include simple animations such as walking on the spot, running or performing grab or reach tasks with the arms, to complex body movements like dances. For all of these animations, we disabled any root movement, meaning that the avatar won't move its origin position. Also, we only use animations, in which the avatar is mostly standing.

We created 4 datasets.

The 22M dataset consists of 22 animations. Like [Zha+18] et al. animations have been flipped in the x-axis, the axis related to left and right of the avatar, and added to the dataset to double the size.

During training multiple neural network architectures, we created bigger datasets. The 62M dataset consists of 62 animations, again with the 62 flipped variations. Playing these animations in Unity it takes 15 minutes. The last dataset 257M comprises 257 animations, with flipped variations. The total play time is 26 minutes.

3.3.6. Recording the Dataset

To record the dataset in Unity, we create a scene with two humanoid avatars present, Groundtruth and Follower. The first avatar is our Groundtruth, and it provides the correct motion, since it is directly animated by Unity's Animator. It plays a sequence of animations during Simulation. The second avatar is the Follower. For each fixed time step, before the PhysX engine performs its calculations, the avatar measures its positional and rotational errors compared to the bones of the animated avatar. Hence, X of the dataset model is recorded. Then, from the calculations in section 3.3.2 and section 3.3.3, for each bone of the Rigidbody avatar, the actuations are computed and y

can be stored. During a recording session, the animated avatar undergoes a series of animations, first the normal animation, then the flipped variant. The Rigidbody avatar follows that movement delayed by one timestep. The session stops when the animator has played all animations. Since we have recordings of X and y for each timestep, the dataset has been recorded.

3.4. Network Architectures

In this section, we cover the different architectures used in this project. In chapter experiments 5, we summarize our comparisons of different architectures in performance.

3.4.1. ActNet

The ActNet has multiple configurations as observable in figure 3.2.

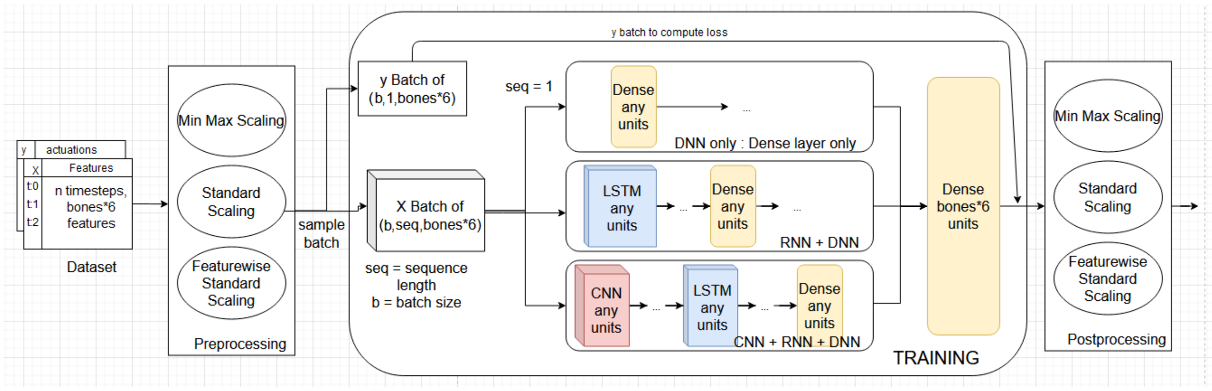


Figure 3.2.: Act Net blueprint for experimentation.

Two Batches are drawn from the dataset, one from X data and one from y data. Input batches might be 3-dimensional, depending on the sequence length. Both batches can be optional preprocessed, more on that in section 3.5. ActNet can have three architecture types. RNN + DNN is a blueprint for recurrent neural networks with LSTM layers, followed by one or more dense layer. DNN only consists of dense layers without any recurrent layer. CNN + RNN + DNN adds convolutional layer in front of the RNN + DNN approach. Finally, all layers are connected to the last dense layer. Note that this layer has linear activation as it is typical for regression networks [Bro20b]. It allows the network to produce outputs in an unlimited range. The output consists of bones * 6 units, so that it fits to the dimensions of the test batch. If the network trained on less bones, for instance on two bones during the overfit experiment 5.2, the network has $b * 6$ output units with $b = 2$. Later data might be post processed.

DNN only

The simplest type of ActNet is the DNN only variation. Once we choose no recurrent layers through hyperparametrization, this network takes 2 dimensional input batches, ergo a sequence length of one. It has no recurrent capabilities but has been used in our experiments. The amount of layers and its units are fully adjustable.

RNN + DNN

With RNN + DNN, we introduce the ability to train from sequences. Training draws batches with n samples and a sequence length of seq creating X-batches in shape of (n, seq, 96), y in shape of (n, 1, 96). Therefore, input batches define their temporal dimension along the first axis. A sequence always consists of consequential time steps. The last timestep matches with a row in the y Batch, which is used during training to calculate the loss. The middle layers are fully dependent on the chosen hyperparameter. Like DNN only, the amount of recurrent layers are selectable by hyperparametrization. Should multiple LSTM layers be chosen, the network keeps sequences between them, while after the last LSTM layer, its output is flattened to fit the dense layer.

CNN + RNN + DNN

We also tested whenever convolutional layer before the LSTM layer can improve prediction performance. Important for the construction of these convolutional layers is that the temporal dimension required for the LSTM layers is not lost. Therefore, we used temporal convolution [ZXL18]. In our case, we move a one dimensional kernel along the temporal dimension before processing the data in LSTM layers.

3.4.2. ActNet : AllForOne

While working with ActNet, we also experiment with a different configuration, which processes the bone data independently. This is different to ActNet, because it has designated input and output notes for specific bones. For instance, the output values of the left bone are corrected only by the left foot bone ground truth. But the function which we used to create y is the same for all bones, which means it is also worth trying a smaller network, which does not train all bones individually, it rather trains one network for each bone. Therefore, the mapping X -> y is trained from the features of all bones in one network. This is the AllForOne idea.

Input

While the input and output features per row of the dataset are bones * 6 values, the data needs to be reshaped for AllForOne. We stack the data along the first axis in slices of 6, causing the network to have 6 input nodes and 6 output notes. Now, all the data is processed regardless of the bone type. It causes the network to learn the function from all samples, but it also loses the ability to learn bone specific information. This idea of training a less monolithic network follows the idea of Grzeszczuk et al. [GTH98]. Since neural networks, like any other machine learning algorithm for classification or prediction, are susceptible to the curse of dimensionality [IM98]. The AllForOne approach aims to tackle the curse of dimensionality problem by reducing the dimensions in input and output drastically.

3.5. Data Pre and Post Processing

In section 2.4.7, we discussed how preprocessing can improve network performance, especially on recurrent neural network models. In figure 3.2 shows preprocessing as an optional step, which transforms the overall X and/or y data first. Then batches can be drawn from it in the training process. We covered MinMax and standard scaling in section 2.4.7. Post processing is necessary if we want to transform the data back into its original scale, which means the previous chosen data transformation is inverted. We need the original scale to make correct statements about the Mean Absolute Error (MAE) with scale that applies to our Unity setting, or if we want to use our network's predictions in Unity.

Feature Scaling

While Standard and MinMax scale every feature independently, which means each row in the dataset is its own feature, with feature scaling we follow a different approach like Grzeszczuk et al. [GTH98]. Full groups of inputs are scaled as one feature rather than having singular features for every row. In our case, we create two standard scales for all of the positional error data and all of the angular error data, resulting in two μ and two σ for X. The same applies for y.

3.6. Ubi Network

The Ubi Network is a Client-Server framework, which connects multiple devices and working environments for research and development [San20]. This server structure can be advantageous for neural network approaches because it allows the network to run on a server with strong hardware, while a client application can use the network's predictions. It also allows a network to train online from multiple sources. Once a network has been trained externally, we connect Unity as Client and Ubi as Server, which then loads the trained network and sends predictions to Unity.

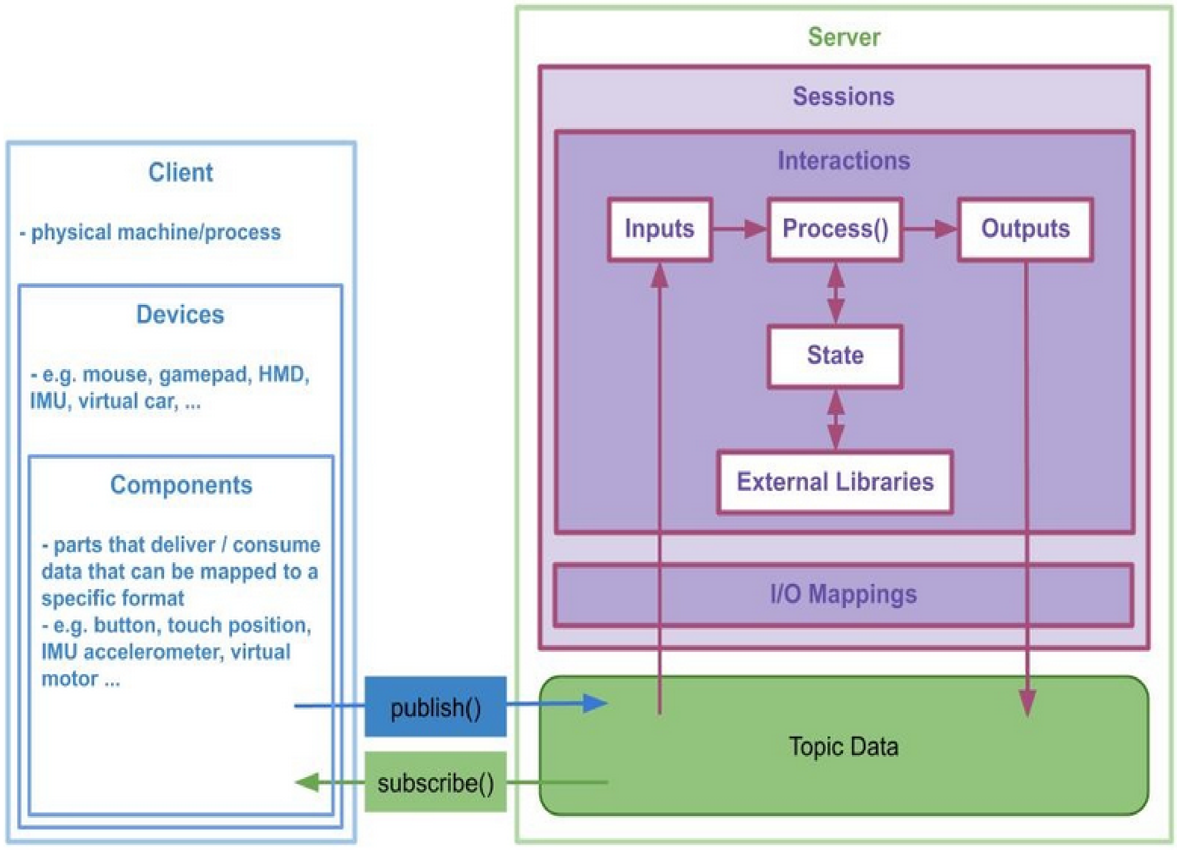

Server
Sessions
Interactions
Inputs
Process()
Outputs
State
External Libraries
I/O Mappings
Topic Data
Client
- physical machine/process
Devices
- e.g. mouse, gamepad, HMD, IMU, virtual car, ...
Components
- parts that deliver / consume data that can be mapped to a specific format
- e.g. button, touch position, IMU accelerometer, virtual motor ...
publish()
subscribe()

Figure 3.3.: Ubi Network Overview [San20]

Figure 3.3 shows how this network framework operates. The Client registers itself on the server by specifying its devices with data structures as components. In our case, the state of the humanoid avatar is captured in a list of float values. These values are published as topic to the Server. A session connects topic data to an Interaction, which processes its inputs repetitively, while maintaining an internal heap of data called state. The outputs of this interaction are stored to a second topic, which the client is subscribed to. Once new data comes in, the client can process new values. So ideally for every timestep, the client publishes data, the interaction takes this data and feeds it to the network. The network makes its predictions and the interaction sends the data back to the client. Since Unity is subscribed to the topic, it can process the values and prepare the Rigidbodies with actuations for the next physics update.

4. Implementation

In this chapter, we explain how our methods have been implemented.

4.1. Overview

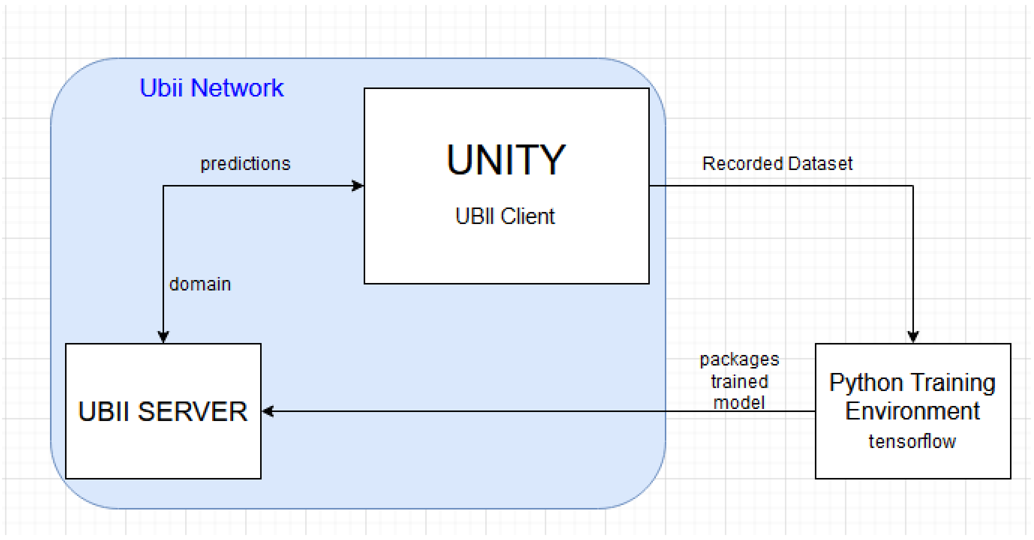


Figure 4.1.: Implementation overview

Figure 4.1 gives an overview to the different environments and how they are related to each other. In the center is Unity, as the simulation environment. There, acts the Rigidbody avatar, which we want to move physically correctly. To the right is a python environment, which contains Tensorflow. Here, neural network architectures are created and trained. Part of the Ubi Network is the backend Server to the left of the figure. Together with Unity as client, they create a Client Server structure. The network loads a Tensorflow instance called model and runs parallel to the Unity client.

4.2. Rigidbody Avatar in Unity

4.2.1. Scene Setup

The scene consists of the GameObjects ExperimentManager, two Rigidbody avatars Groundtruth and Follower, such as the optional GameObject UbiiClient. Each avatar contains its surface and joint GameObjects, as well as the armature representing the hierarchical bone structure by parented GameObjects. Each avatar has the Rigidbody Behaviour attached. Only the Groundtruth has an Animator attached.
To create human like motions, the Groundtruth is leading by being animated from the Animator component. How and in which order animations are played is designed by creating an Animation Controller which is passed to the Animator. For all the different datasets variations, we created one Animation Controller, hence the different animation sequences can be changed before runtime. When play mode starts, the Groundtruth moves through the animation clips sequenced by the Animation Controller. The Groundtruth stands in the origin of the scene, where the second Rigidbody avatar, the Follower, is located, too. The Follower doesn't have an animator, but it holds the Recorder Behaviour and the Move along Behaviour.

4.2.2. GameObject: ExperimentManager

The ExperimentManager is a GameObject in the scene, which is used to establish communication between important GameObjects and their Behaviours. It has a Behaviour named like the object, which is implemented as singleton. It exists only once in the scene and it is accessible by all classes. It finds other Behaviours in the scene and distributes them.

4.2.3. Behaviour: RigidbodyAvatar

This Behaviour is attached to both Rigidbody avatars. Its purpose is to attach the rigidbody components to the armature and to apply rigidbody settings and colliders. It also establishes the required object hierarchy.

In figure 4.2, you can see how the script moves the surface objects into the armature. This is necessary, because while we attach a rigidbody on each bone in the armature, the surface objects represent the actual physical shapes. If we keep the surfaces only as child of the root object, the surfaces will stay in t-pose position, regardless of any animation happening.

The figure shows to the right, surface objects as child of root. To the left, we can see that the surfaces move with the animation, as soon as the surface objects are moved

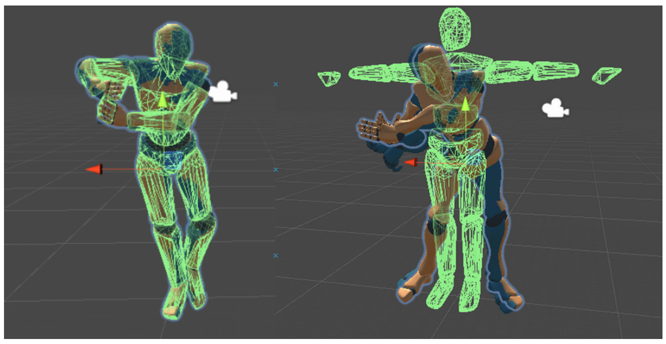


Figure 4.2.: On the left: Surfaces are parented correctly to the bone hierarchy. On the right: Surfaces stay in t-pose unparented.

into the bone hierarchy.

Function: Start

On Start, the Behaviour looks for an animator component on its root. If found, it knows it's the Groundtruth avatar, if not it's the Follower. Then, it starts the Coroutine SetListeners.

Coroutine: SetListeners

Coroutines allow the execution of commands to wait for a while before continuing. The Coroutine SetListeners waits for 0.2 seconds, then it creates the UnityEvent AvatarReady. With the ExperimentManager, the listening Behaviours RigidbodyRecorder andRigidbodyMoveAlong are added. When RunInitialization is set to true, the Rigidbody is initialized with the correct parenting. To set up the structure we need, we iterate over the objects in the armature hierarchy. Each avatar bone name has the following pattern: <AvatarBonePrefix>_<bonelocation>. With this name, we can find the surface named <AvatarSurfacePrefix>_<bonelocation> and make it a child of the bone. The bones get a Rigidbody attached with the following attributes. Gravity is disabled. Angular Drag is set to zero, so that the Rigidbody does not slow down on its own. The body is set dynamic or kinematic based on the isDynamic flag. Usually, isDynamic is true so that the avatar responds to collisions with other objects. Is setRBOriginToArmature set to true, the origins of the avatar's rigidbodies are set to the transform position of the bone. This setting did not impact the recording or the performance of tested networks. From the mesh of the surface object, the mesh collider is created and set to convex and is inflated, if the rigidbody is set to dynamic. For dynamic Rigidbodies, convex meshes are required. When the Rigidbodies are constructed, the Unity Event AvatarReady invokes its listeners RigidbodyRecorder.PrepareRecorder and Rigidbody-MoveAlong.PrepareFollow. These listeners are functions of their respective classes and start executing once invoked.

Function: Update

The Update function executes every frame and checks if this instance has an animator. This is the case for the Groundtruth avatar. It checks if the Animation Controller reaches a state named 'finish'. While every animation in the Animation Controller has it's own state name, finish is a last state without animation clip. Once this state is reached, we set the Play Mode to false. Then, the simulation stops, which ensures that there are no time steps recorded without motion.

4.2.4. Behaviour: RigidbodyMoveAlong

This Behaviour is only used on the Follower. There are three modes implemented. Default is the arithmetic following, in which the Rigidbodies actuations are inferred directly by dividing the positional and rotational distance to the Groundtruth Rigid-bodies by time. The second mode is via file, the class reads row by row from csv input. The last mode is using the Behaviour Ubii Tensorflow Communication to post data for prediction and listen to data to process. If resetVelocityCalculation is set to true, during actuation the rigidbody resets all previous velocities on itself. The Follower has a dictionary, which maps each bone to the Groundtruth target bone, e.g Follower_hip will map to Groundtruth_hip.

Function: Start

On Start, the dictionary is empty initialized. The Unity Action MakeArmatureMap adds the functions MapArmature and FetchRecorder to itself. The action waits till AvatarReady invokes it, then the added functions are executed. If a Ubii Tensorflow Communication instance is in the scene, it is found. As the name suggests, it is used to communicate to the server if the client needs to. If we specified the file reading mode, the Behaviour opens a file with a StreamReader.

Function: OnApplicationQuit

On stopping play mode, we close files opened during start.

Unity Action : PrepareFollow

PrepareFollow waits for MakeArmatureMap to execute it. First, it maps the armature by creating a one to one mapping from Follower bone to target Groundtruth bone. It iterates over the children in the armature of the Groundtruth avatar, for each it finds the corresponding Follower bone by name. This pair of bones are added as entry to the dictionary (<followerbone, targetbone>). After that, the second subaction of PrepareFollow fetches a reference to the recorder with the ExperimentManager. With the dictionary ready and the recorder fetched, Move Along is ready to control the Follower and to pass values to the Recorder if there is the need to record a dataset.

Function: FixedUpdate

In its FixedUpdate, the Behaviour checks if the dictionary has been initialized. If that is the case, the current time since simulation start is stored in the variable time. The time's value is also shared with the recorder. Then, based on flags set, it goes into one of the three different modi.

Actuate Rigidbody by Calculation

This mode is used to record the dataset. For each Rigidbody pair, the distances from Follower bone to target bone in position and rotation are measured.

```
Vector3 positionDelta = RigidbodyErrorPos(source.transform, target);

public static Vector3 RigidbodyErrorPos(Transform bone, Transform target)
{
       return target.position - bone.position;
}
```

The distance vector between two positions is the subtraction of target minus current position.

```
Vector3 angularDelta = RigidbodyErrorAngles(source.transform, target);

Vector3 a = bone.rotation.eulerAngles;
Vector3 b = target.rotation.eulerAngles;
{
       return new Vector3(Mathf.DeltaAngle(a.x, b.x), Mathf.DeltaAngle(a.y, b.y),
       Mathf.DeltaAngle(a.z, b.z));
}
```

The distances in degrees are the smallest angles between the euler angles of target and source. Note, these angles are in degrees. We also tested recording rotational errors in radians which resulted in no meaningful effects.

Positional and rotational error plus time step value is sent to the recorder. It will write a new row in the dataset input file. Next, the Behaviour calculates the inputs of the Rigidbody functions AddForce and AddTorque to reduce the distances in the next physics update, based on the calculations in sections 3.3.2 and 3.3.3.

```
Vector3 velo = positionDelta / Time.deltaTime;

public static void ApplyForce(Vector3 velo, Rigidbody actuated, bool reset)
{
if(reset)
{
actuated.velocity = Vector3.zero;
}
Vector3 prefVelo = actuated.velocity;
Vector3 newVelocity = (velo - prefVelo);
actuated.AddForce(newVelocity / Time.deltaTime, ForceMode.Acceleration);
}
```

Force is applied to the actuated rigidbody component, by calculating the required velocity from the total velocity to close the positional error. Notice how the resetVelocityCalculation mode resets the Rigidbody, if enabled. This mode has been used in earlier stages of the project, but resetting the rigidbody each timestep kills external forces which act on it. Since the avatar should be moved physically correctly, this mode is avoided and is legacy. With the new velocity, we can calculate the acceleration required to push the Rigidbody to its target destination by dividing velocity with time. Notice how we use the ForceMode Acceleration to remove the complexity of masses. Whereas ForceMode.Force requires an actual force, ForceMode.Acceleration will accelerate the Rigidbody by its first parameter in regard to its own weight. Next, we observe Torque actuation.

```
Quaternion currentRotation = source.rotation;
Quaternion rotationDir = target.rotation * Quaternion.Inverse(currentRotation);

float angleInDegrees;
Vector3 rotationAxis;
rotationDir.ToAngleAxis(out angleInDegrees, out rotationAxis);
rotationAxis.Normalize();

Vector3 angularDifference = rotationAxis * angleInDegrees * Mathf.Deg2Rad;
Vector3 angularSpeed = angularDifference / Time.deltaTime;

if(resetVelocityCalculation)
{
       angularSpeed = new Vector3(rotationDir.x, rotationDir.y, rotationDir.z);

}
ApplyTorque(angularSpeed, source, resetVelocityCalculation);
```

To find the Torque, which is required to move the Rigidbody from its source rotation to a target rotation, the Behaviour finds the rotation direction by multiplying the target rotation quaternion with the inverse of its current rotation. This rotation direction is converted to a rotation in euler axes. It's normalized to ensure a vector with a length of one. The Function ToAngleAxis returns the angle, which is required to reach the target rotation. With angle and rotation axis, we can find the angular Speed required to rotate the Rigidbody to its target rotation. Note how the angle in degrees is converted to radians. AddTorque requires radians input.

```
public static void ApplyTorque(Vector3 anguV, Rigidbody actuator, bool reset)
{
        if(reset)
        {
                actuator.angularVelocity = Vector3.zero;
        }
        Vector3 anguVelo = anguV - actuator.angularVelocity;
        if (reset)
                actuator.AddTorque(anguVelo / Time.deltaTime, ForceMode.Force);
        else
        {
                actuator.AddTorque(anguVelo / Time.deltaTime, ForceMode.Acceleration);
        }
}
```

Again, if resetVelocityCalculation is true, we reset the Rigidbody's acting velocities, in this case the angular ones. We compute the new angular velocity based on acting angular velocity. Then, we apply this velocity as acceleration. This is important, because AddTorque will accelerate the Rigidbody while taking its Inertia Tensor into account. In the case of resetting all previous angular velocity, we can apply its force directly. As we discussed earlier, this code is legacy. The velocity and angular velocity passed to ApplyForce and ApplyTorque are also passed to the recorder, so that these values are stored in the dataset output file.

These calculations are done for every Rigidbody pair. In the next PhysX cycle, the follower's Rigidbodies move according to their new active velocities. These calculations are tested during playtime. Since the Groundtruth avatar demonstrates body movements, the correctness of these calculations can be seen subjectively during the recording process as the other avatar follows the movement with all of its Rigidbodies with a tiny delay.

Actuate Rigidbody by Tensorflow Server

If the global flag useTensorflow is set to true, the avatar is actuated by data sent from the server. It receives a row of data from the Tensorflow Ubii Communication Behaviour and processes it.

```
void ProcessData(List<float> receiveData)
{
toFollowAvatar.StartAnimator();
if (receiveData.Count == 0)
return;
float tfTime = receiveData[0]; ...
```

Note how the animator of our Groundtruth avatar is activated, hence, it starts to move. This lets the Groundtruth wait until the Client Server communication has been established. tfTime stores the time step the prediction has been made for. Next, we see multiple reasons to exit this processing.

```
... int idx = 1;
foreach (KeyValuePair<Transform, Transform> pair in armatureMap)
       {
       if (idx + 5 > receiveData.Count)
       {
              Debug.LogError("Received data from tensorflow has not enough values");
              return;
       }
       if (tfTime <= lastTFProcessed)
       {
              Debug.LogError("Prcessing old values, abort ");
              return;
       }
       if(pair.Key.name != pair.Value.name)
       {
              Debug.LogError("Wrong key value pair in armaturemap, abort ");
              return;
       }
}
```

Reasons to stop execution are, if the received data doesn't have enough values, old data is again processed, mostly because of client server lag, or if the dictionary isn't mapped correctly. The loop iterates over the bones in the dictionary. Per iteration, the values in the received data are processed in blocks of six:
[bone_newVelocity_x, bone_newVelocity_y, bone_newVelocity_z, bone_newAngularVelocity_x, bone_newAngularVelocity_y, bone_newAngularVelocity_z].

```
... Vector3 velo = new Vector3 { x = receiveData[idx], y = receiveData[idx + 1],
 z = receiveData[idx + 2] };
ApplyForce(velo, pair.Key.GetComponent<Rigidbody>(),resetVelocityCalculation);
idx += 3;
Vector3 anVe = new Vector3 { x = receiveData[idx], y = receiveData[idx + 1],
 z = receiveData[idx + 2] };
ApplyTorque(anVe, pair.Key.GetComponent<Rigidbody>(), resetVelocityCalculation);
idx += 3;
}
lastTFProcessed = tfTime;
} //End of Function ProcessData()
```

Then, the data can be passed to ApplyForce and ApplyTorque, so that the Rigidbodies are moved based on the values the network has predicted.

Actuate Rigidbody by File

If usePredictionFile is true and a prediction file is opened by the StreamReader, each line is handled as a row of predictions for one time step. This datafile has to be a csv file, with the rows as prediction per time step, and the columns with features.

4.3. Server Implementation

The server is Notejs based. On start up, it loads classes such as the TensorflowEvaluateService, which wraps the functionality to load a Tensorflow model and to provide predictions on runtime.

4.3.1. Class: TensorflowEvaluateService

The class constructor requires certain files to be prepared to the tfdata folder of the server's root directory.

```
this.pathModel = "file://tfdata/model.json";
this.minMaxFile = "./tfdata/minMax.txt";
this.standardFile = "./tfdata/standardScale.txt";
this.standardFeatureFile = "./tfdata/standardFeaturedScale.txt"
```

The model.json file contains meta information to load a fully trained Tensorflow network. The weights are stored in shard files which are indexed by the json file. The other text files contain values necessary to apply pre and post scaling. These files are created, just like the model, during training sessions.

Function: loadModelMinMax

There are three different ways to initialize the model, since we experimented with different pre and post scaling options. First, there is MinMax scaling, which requires minimum and maximum values per feature. We load these from the datapath specified in the constructor, and store the values as member classes.

```
async loadModelMinMax (allForOne)

        console.log(this.pathModel);
        console.log(this.minMaxFile);
        try {
                this.AllForOne = allForOne;
                this.model = await tf.loadLayersModel(this.pathModel, {strict: true});
                var filetext=fs.readFileSync(path.resolve(this.minMaxFile), 'utf8').trim();
                this.layerInputShape = this.model.layers[0].batchInputShape;
                this.layerInputShape[0] = 1;
                filetext = filetext.replace("[","");
                var fileParts = filetext.split("]");
                this.scalerXMin = (fileParts[0].split(",")).map(Number);
                this.scalerXMax = (fileParts[1].replace("[","").split(",")).map(Number);
                this.scalerYMin = (fileParts[2].replace("[","").split(",")).map(Number);
                this.scalerYMax = (fileParts[3].replace("[","").split(",")).map(Number);
                this.mode = "MinMax"
        } catch(e) {
                console.log('Error while initializing TensorflowEvaluateService:', e.stack);
        }
        return this.model;
```

With the boolean parameter allForOne it is specified if the model is a AllForOne model. This is important, then the data comes in, because the data for an AllForOne model needs to be reshaped. To note is that the model is loaded with the parameter strict: true. It enforces that the trained weights are loaded in the exact shape they have been trained. This is very important to guarantee that the network is created exactly like it was saved. The scaler values are loaded from the file specified by variable minMaxFile. This file contains in the first row all minimum values of all input features. In the second row, the maximum values of all input features. In the third and final row the min and max values of the output features.

Function: loadModelStandard

The only difference to LoadModelMinMax is that a flag this.mode is set to use standardScaling. And it loads mean and standard values from the file specified from this.standardFile.

Function: loadModelFeatureScaled

The same as loadModelStandard, with the exception that it loads mean and standard derivation of the two features of input and output:

```
...
var fileParts = filetext.split("]");
var meanXPos = (Number(fileParts[0]));
var varXPos = (Number(fileParts[1]));
var scaleXPos = (Number(fileParts[2]));
var meanXAng = (Number(fileParts[3]));
var varXAng = (Number(fileParts[4]));
var scaleXAng = (Number(fileParts[5]));
var meanYPos = (Number(fileParts[6]));
var varYPos = (Number(fileParts[7]));
var scaleYPos = (Number(fileParts[8]));
var meanYAng = (Number(fileParts[9]));
var varYAng = (Number(fileParts[10]));
var scaleYAng = (Number(fileParts[11]));
this.meanX = [meanXPos, meanXPos, meanXPos, meanXAng, meanXAng, meanXAng];
this.scaleX = [scaleXPos, scaleXPos, scaleXPos, scaleXAng, scaleXAng, scaleXAng];
this.meanY = [meanYPos, meanYPos, meanYPos, meanYAng, meanYAng, meanYAng];
this.scaleY = [scaleYPos, scaleYPos, scaleYPos, scaleYAng, scaleYAng, scaleYAng];
this.mode = "standardFeature"
...
```

Note, that we used a trick here to construct the member classes. Data to process comes in as a vector with a length of number of bones * 6 values. We use that in the predict function.

Function: predict

The Function predict scales an incoming input array based on how the model has been loaded. The different implementations of the scaling operations can be found in the appendix C.1. First the incoming data is scaled, then the data is feed to the network. The network returns a prediction which is still scaled, therefore the scaling operation is inverted and the data is returned. Is this.AllForOne set to true, the data must be reshaped first. The regular ActNet trains 16 bones which results in 16*6 = 96 features. AllForOne processes the data of each bone independent, therefore we split the input into 16 batches and run the network on those. This happens in the function predictAllForOne. Besides of that reshape, it does the same as the regular predict, except that the 16 outputs of AllForOne models are joined together to one array before they are returned.

4.3.2. Interaction: TensorflowEvaluateService

This Ubi Interaction is created once the first topic data comes in, Thus, OnCreate is called. It is going to process

Function: onCreate

OnCreate calls loadModelMinMax, loadModelStandard or loadModelFeatureScaled, depending on if the flags for standardScale or featureStandardScale are set, the default is MinMax. The flag AllForOne is passed as parameter to the load functions. Timestep of the last processed prediction is set to -1, then the overall interaction is set to be ready for processing.

Function: process

```
(inputs, outputs, state) => {
        if(state.ready)
        {
                if(inputs.TensorflowInput !== undefined)
                {
                        var inputArray = Array.from(inputs.TensorflowInput.elements);
                        var timestep = state.modules.tes.fetchTimeStep(inputArray);
                        state.modules.tes.spliceinputData(inputArray);
                        if(timestep > state.timestampLastPrediction)
                        {
                                state.timestampLastPrediction = timestep;
                                var temp = state.modules.tes.predict(inputArray);
                                temp.unshift(timestep);
                                outputs.TensorflowOutput = { elements: temp};
                        }
                }
        }
}
```

Inputs and outputs make up the data, which is sent by the topics between server and client. State is the Interaction's scope, which persists over different process calls and OnCreate. If OnCreate is finished and the state.ready is set to true, we fetch a timestep from input data. Only if this timestep is newer than previous processed time steps, execution is continued. This prevent the server to re-predict values. Then, we use the TensorflowEvalucateService (TES) class to predict the data, the current timestep is attached in front of the prediction and overrides lastPrediction. Then the overall data array is set as interaction output.

4.4. Client Implementation

To connect Unity with the Ubi Framework, the UbiiClient class connects Unity with the Ubi Network, based on Socket technology with NetMQ [Net]. Functions provided by UbiiClient allow other Behaviours to request services, which are necessary to set up the communication pattern we described in section 3.6.

4.4.1. Behaviour: Tensorflow Ubii Communication

To connect Unity with the TensorflowEvaluateService of the server TensorflowUbiiCommunication creates a Session, which starts the process running the tensorflow module for predictions (4.3.2). It also manages the two topics, subscribe and publish. These are defined as float lists.

Function: Start

Once the function Start begins, the Behaviour finds a UnityClient in the scene and once found, it prepares communication to the server. First, it awaits if the UbiiClient is connected to the UbiiServer, it delays its progress by one second, then the previous Topic is initialized and DeviceSpecs are created. The deviceSpecs are constructed with a client ID, the devicetype Participant and the two topics, publisher and subscription.

```
Ubii.Services.ServiceReply deviceRegistrationReply =
 await ubiiClient.CallService(new Ubii.Services.ServiceRequest
      {
      Topic = UbiiConstants.Instance.DEFAULT_TOPICS.SERVICES.DEVICE_REGISTRATION,
      Device = ubiiDevice
      });
      if (deviceRegistrationReply.Device != null)
      {
             ubiiDevice = deviceRegistrationReply.Device;
      }
```

Next, we register a device with the deviceSpecs ubiiDevice on the server. Notice, how the server returns a device object with its reply, once registration was successful.

```
Ubii.Services.ServiceReply interactionReply =
await ubiiClient.CallService(new Ubii.Services.ServiceRequest
       {
       Topic = UbiiConstants.Instance.DEFAULT_TOPICS.SERVICES.INTERACTION_DATABASE_GET,
       Interaction = new Ubii.Interactions.Interaction { Id = interactionID }
       });
       if (interactionReply.Interaction != null)
       {
              ubiiInteraction = interactionReply.Interaction;
       }
```

Next, we query from the database the Interaction ID which specifies the TensorflowEvaluateService Interaction from section 4.3.2. To do so, a second server request queries the server's database to check if the Interaction ID is contained, the server responds with an Interaction object on success.

```
private void MakeUbiiSession()
{
       ubiiSession = new Ubii.Sessions.Session
       {
              Name = "TestTensorflowConnection - Session IO",
              Description = "Testing Session",
              ProcessMode = Ubii.Sessions.ProcessMode.IndividualProcessFrequencies
       };
       ubiiSession.Interactions.Add(ubiiInteraction);

       inputMapping = new Ubii.Sessions.InteractionInputMapping
       { Name = input.Name, Topic = input.Topic };
       outputMapping = new Ubii.Sessions.InteractionOutputMapping
       { Name = output.Name, Topic = output.Topic };
       IOMapping = new Ubii.Sessions.IOMapping{InteractionId = this.ubiiInteraction.Id};
       IOMapping.InputMappings.Add(inputMapping);
       IOMapping.OutputMappings.Add(outputMapping);
       ubiiSession.IoMappings.Add(IOMapping);
}
```

Next, an UbiiSession object is initialized by adding the previous acquired interaction. Notice how the Topics design Input and Output mapping of the interaction. Input.Topic is the Publish topic, Output.Topic is the Subscription Topic. This session is started similarly to previous Device registration, with the difference that the client calls a service with a different service request:

```
Ubii.Services.ServiceReply sessionRequest =
await ubiiClient.CallService(new Ubii.Services.ServiceRequest
{
       Topic = UbiiConstants.Instance.DEFAULT_TOPICS.SERVICES.SESSION_RUNTIME_START,
       Session = ubiiSession
});
```

Should the server respond with a reply containing the session object, the session is running and awaits publish data.
Finally, the communication Behaviour subscribes to the subscription topic

```
await ubiiClient.Subscribe(output.Topic, (Ubii.TopicData.TopicDataRecord record) =>
{
output.Data = record.FloatList;
});
running = true;
```

Upon receiving data from the server, output.Data is set. This data is eventually processed by the Follower Avatar.

Update

For each frame the Behaviour checks if the publish topic has a time step which is newer then the timestep, which was published before. If that is the case, it publishes the topic with new avatar data to the server and updates the lastPublished timestep.

OnDisable and OnApplicationQuit

Once the application closes or this behaviour is disabled, it calls services to stop the session first and to deregister the client.

4.5. Tensorflow Training Environment

Neural Network training is running in a python environment having Tensorflow installed such as numpy, sklearn, and pickle.

4.5.1. Script: ActNet.py

ActNet.py is a script which performs one training session. Given a set of hyperparameters, it loads the data and iterates through all combinations of hyperparameter elements. For each selection of hyperparameter, it builds the model, trains it a fixed amount of steps, and saves the model with the best test performance. For the whole process, a log file is created. In chapter 5, we compare these architectures against each other to see which is working the best.

Function: build_model

Given a selection of hyperparameters, this function creates a model.

The sequential Tensorflow model is created based on the passed parameters. The variable cnn_units is an array of integer, specifying how many convolutional layer are to be built and how many units they have. For each integer element in cnn_units array, the model creates convolutional layer. The integer value specifies the amount of units. Optional regularizer can be specified. These settings are documented in the training logs. See the appendix B for all documented training runs.

Next, recurrent layers with LSTM modules are created for each array element in rnn_units array, with the element's value as an amount of units. The algorithm takes into account if the next layer to be added is not the last recurrent layer. If that is the case, it sets an option on the LSTM layer to return the results as sequence. This maintains the temporal axis for the next recurrent layer. If not, outputs are reduced to the units of the last layer, without sequence. Activation for these are default specified by Keras' LSTM implementation [Ker], with recurrent activation as sigmoid.

Next, dense layer are created from the dnn_units array like the recurrent layer. All configurations end with the last output layer, containing the number of trained bones times 6 values, respectively. For regression networks, a linear activation for the last layer is important (see 3.4.1). In training sessions with regularization, we added batch normalization only between dense layers.

Data load and preprocessing

Two data files are loaded based on the path specified in the script. A Dataset comes with one file of Input data and one file of output data. Both are csv files. The whole set is loaded in a 2d numpy array. While reading the rows, the first header row is discarded. For each line, the first value is skipped too, since it stores the timestep.

Based on argument flags a scaling method is used. The different scaling techniques are discussed in section 3.5. The scalers are functions of the sklearn.preprocessing package. As input, they take 2 dimensional arrays, whereas the rows are entries and the columns are features. Each feature is independently scaled, meaning that each row gets its own min and max value, or its mean and standard derivation.

For featured scaling it is required to reshape the data. Therefore, input and output data is split between positional data and rotational data. Then the split 2d arrays are reshaped to be one dimensional. Then scaling is applied. And then the data is joined back together and reshaped to be in its original two dimensional form. For these transformations, numpy is used. Regardless of the preprocessing, the data is split into 80% of training data, the remaining 20% of test data.

Function: get_batch

get_batch returns the two batches required for supervised training.

```
def get_batch(dataX, dataY, seq_length, batch_size):
        n = dataX.shape[0] - 1
        idx = np.random.choice(n, batch_size)
        input_batch = []
        output_batch = []
        for x in range(batch_size):
                while idx[x] - seq_length < 0:
                        idx[x] += 1 #just take the next
                input_batch.append(dataX[np.arange(idx[x]-seq_length, idx[x], dtype=int)])
                output_batch.append(dataY[idx[x]])

        x_batch = np.reshape(input_batch, [batch_size, seq_length, dataX.shape[1]])
        if(cnn_units is not None):
                x_batch = np.reshape(x_batch, [batch_size,seq_length,dataX.shape[1]])
        y_batch = np.reshape(output_batch, [batch_size, dataY.shape[1]])
        return x_batch, y_batch
```

For each batch element the algorithm selects an index of a random row in the dataset. Is $idx - seq_length < 0$, in words, is the index minus the sequence length smaller then 0, we increase the index so that the sequence frame is not ranging into negative indices. The batch of X is created by appending the rows of the sequence together. This is done for every batch element. For y, no sequences are necessary, since we want to predict the actuations of the last time step from the X batch.

Training Process

Executing the training script once, for each hyperparameter combination which are parametrized, the network creates a Tensorflow model with the build_model function and sets up the optimizer. The optimizer uses Tensorflow's Adam [Ten20] implementation. For each model, the training process runs for a limited amount of iterations For each step the algorithm performs a train step and a test step. The train step takes an input and output batch from the training data, computes a prediction and loss, returns its gradients and passes these gradients to the optimizer for backpropagation. The test step takes data from the test data and returns prediction and loss on it.

```
if np.mean(loss_test) < compareLoss:
        compareLoss = np.mean(loss_test)
        [...]
        model.save(model_prefix)
```

During training, the best performing test loss is tracked. Once any training step scores a test loss below compareLoss, compareLoss is updated and the model with its current weights is saved. Note that the lowest test loss in compareLoss is tracked through all models. Each model undergoes the same number of training iterations, then, the best performing model is re-loaded and stored to json format, which can be loaded in a js environment.

4.6. Collider Dome

With the goal to record a dataset containing collisions, the Collider Dome has been implemented. The Collider Dome is a hemisphere with fully adjustable radius. While the avatar is located in the center of the hemisphere, on simulation start the collider dome spawns a set of points randomly on the surface of that hemisphere. During runtime, while the standard recording process is running, the collider dome spawns one box as collider, which chooses a random target from a set of selected follower bones as target. The box moves towards it's target causing a collision. The collider dome is implemented as optional GameObject which can be toggled on and off.

Figure 4.3 shows the red spawn points around the two avatars. The box itself is visible by green outlines. In the figure, it pushes into the avatar from the right. It is visible how the collisions only happen with the Follower (in orange).

4.6.1. Behaviour: Collider Dome

Function: Start

On start, the current scale of the GameObject is stored as global variable domeSize. Then the spawn points for the box collider are generated. To do so, we follow the implementation of Dammertz [Dam12], who samples from a uniform hemisphere with hammersley vectors. These points are scaled to fit domeSize and for each surface point a red sphere is instantiated.

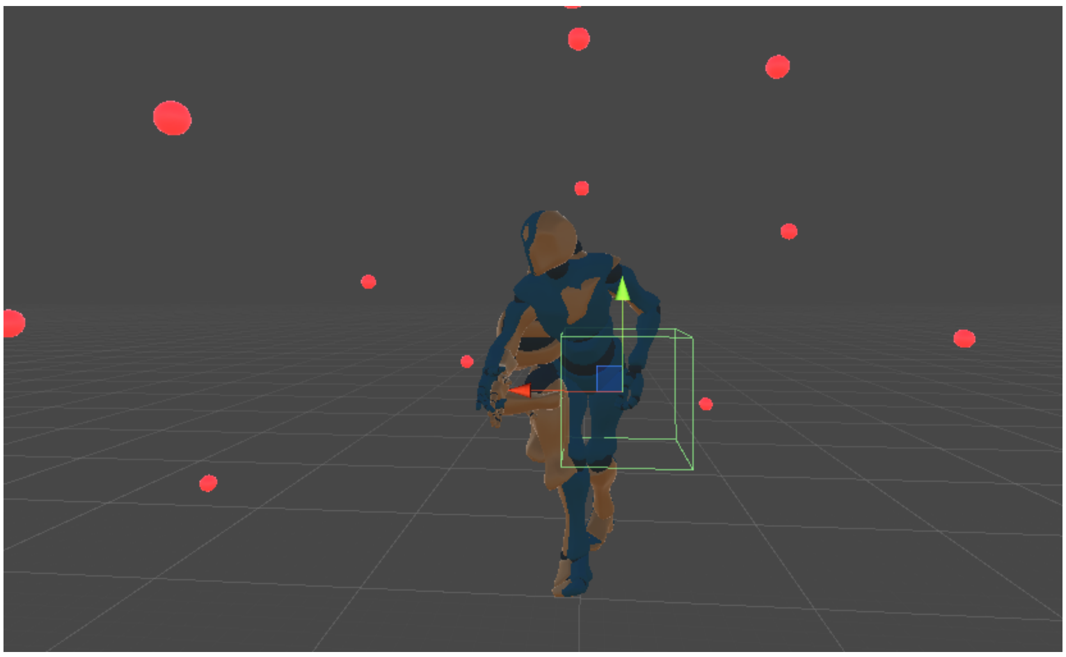


Figure 4.3.: The Collision Dome produces collisions by moving box colliders into the recording scene.

Function: FixedUpdate

In the FixedUpdate loop, if there is no active collider we instantiate a new collider box on one of the hemisphere points. As the collider spawns, we add Force to it, directing it to one of the target bones. With every spawn, a different target is selected. Once the collider is on its way, a timer will count down to timeToBeAlive seconds. As soon as the collider exists longer, the collider is deleted and in the next FixedUpdate a new collider spawns.

4.7. Implementation Limitations

During development, we noticed that this conversion of tensorflow models to json files has problems with keras regularizer. Weights regularizer "L1" and "L2" cause the load process on the server to fail. It is possible to simply unset those regularizers by setting the values in the json file to null, but it is uncertain if this affects the performance of predictions. The json model builder produces another error, which needs to be mentioned. Using LSTM layers from keras, the converter names lstm layer "lstm", while also appending "lstm/lstm_cell/kernel", "lstm/lstm_cell/recurrent_kernel" and "lstm/lstm_cell/bias" to the model. Once the server is running this model, it fails because the layer "lstm" does not find its elements. To fix this issue, lstm_cell needs to be removed from these three object names. For instance rename "lstm/lstm_cell/kernel" the kernel to "lstm/kernel".

5. Experiments

5.1. Data Analysis

In this section, the recorded datasets are analysed.

5.1.1. 62M

The 62M dataset contains the recording of 62 animations, plus its in x-Axis mirrored variations. Let's observe the features mixamorig HipsErrorPos x and mixamorig HipsErrorAng x in figure 5.1 and figure 5.2.

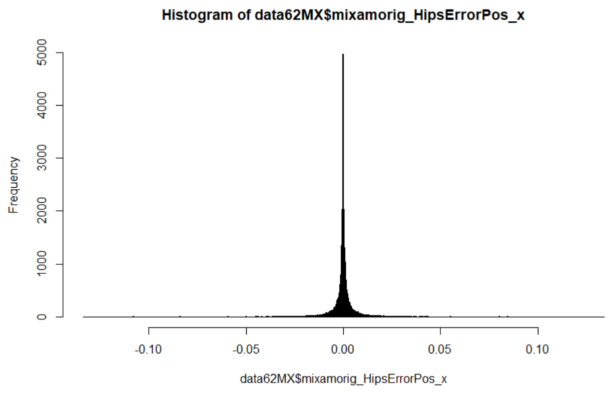


Figure 5.1.: 62M HipsErrorPos x

In figure 5.1, we observe a histogram depicting the frequency of occurring values for the positional error in x of the hip bone. We identify a zero centred distribution with a very steep trend. From the overall shape, we can identify a normal distribution with a small derivation around 0.00909328. Minimum value of this feature is -1.334e-01, maximum value is 1.339e-01.

In figure 5.2, we observe a histogram depicting the frequency of occurring values for the angular error in x of the hip bone. We identify a zero centred distribution with a very steep trend. From the overall shape, we can identify a normal distribution

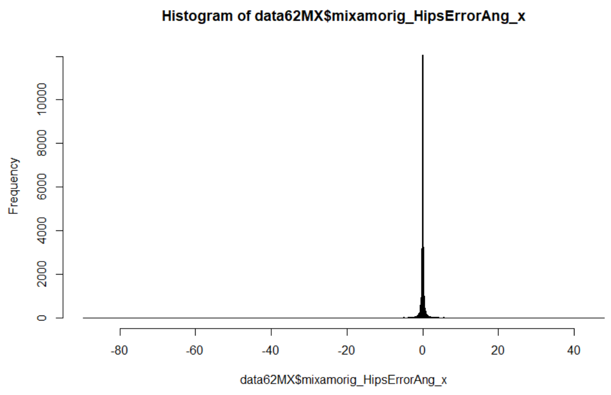


Figure 5.2.: 62M HipsErrorAng x

with a small derivation around 2.294059. Minimum value of this feature is -89.47397, maximum value is 47.74844.

The normal distribution is observable for all of the features in the dataset. More examples can be found in the appendix A.1.

5.1.2. 257MR

The 257MR dataset is a re-recording of the original 257M dataset, which consists of 257 animations, plus it's in x-Axis mirrored variations. During experimentations, we notice that the recorder failed to record some timesteps correctly and left some rows with zero values. The 257MR dataset has been recorded in two play sessions and manually joined together. Again, we observe the features mixamorig HipsErrorPos x and mixamorig HipsErrorAng x in figure 5.3 and figure 5.4.

In figure 5.3, we observe a zero centred distribution, again with a step trend. From the overall shape, we can identify a normal distribution with a small derivation around 0.01484116. A Less steep distribution than the previous 62M dataset on the same feature. With samples recorded from more animations, this feature distribution seems to be more spread out. The shape of the normal distribution is better visible here. The standard derivation is bigger, whereas the borders for minimum and maximum are less impacted.

In figure 5.4 we observe a histogram depicting the frequency of occurring values for the angular error in x of the hip bone. We identify a zero centred distribution, with a very step trend. From the overall shape, we can identify a normal distribution with a standard derivation around 2.294059. Minimum value of this feature is -89.71252, maximum value is 124.90970. Compared to the previous dataset, this dataset seems

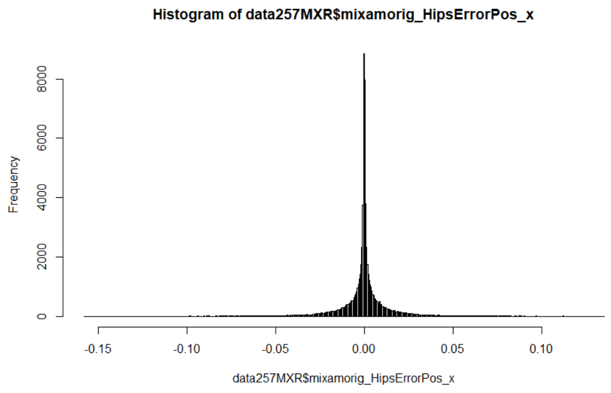


Figure 5.3.: 257MR HipsErrorPos x

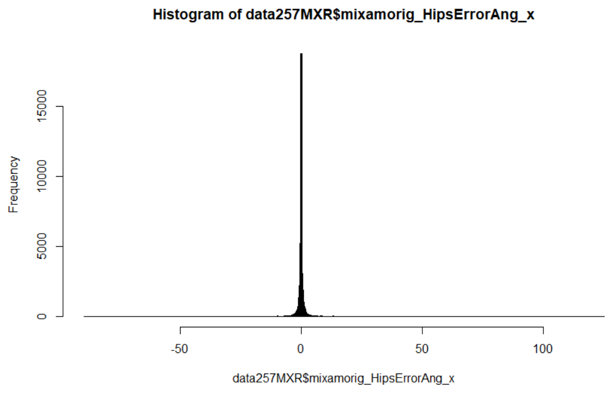


Figure 5.4.: 257MR HipsErrorAng x

to have much more dynamic animations, indicated by the bigger range of values. The normal distribution is visible for all the features in 257MR, more examples are provided in the appendix A.7

5.1.3. All For One 257R

To ensure the normal distribution of our data for the All For One approach in section 3.4.2, we also analysed the data, regardless of its bone type.

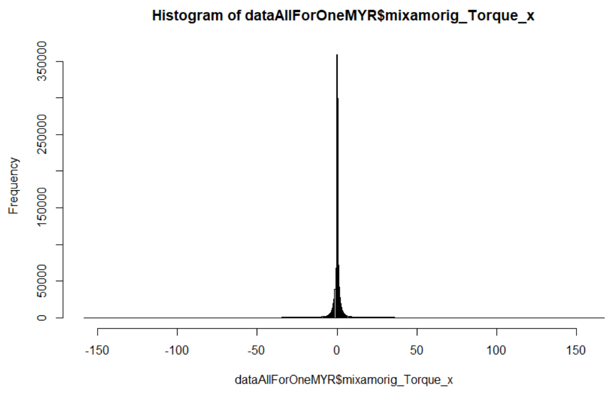


Figure 5.5.: 257AllForOne Torque x

In figure 5.5, we can observe normal distributed data for the Torque along the x-axis across all bones. In the appendix A.13 and A.14, the histograms for all six features show a normal distributed shape, similar to the previous plots. In total we come to the conclusions that standard scaling should be applied on X and y. This is also covered by our findings in section 5.3.1.

5.2. Overfit Experiment

First, we explored if it's possible to overfit a very simple motion. The T2Bones dataset is a small recording of a walking animation. In this dataset, only the hip and the left upper leg have been recorded. The hip does move very slightly, the leg swings back and forth. In this experiment, we aimed for a strong performance in training loss.

```
best result archieved with : Saving weights with lr=0.001, batchsize=128,
 seq_length=5, dnn=[2048, 2048], rnn=[128, 128], iter=1808, loss=0.0003781
```

The best result has been achieved with an ActNet consisting of two Long Short-Term Memory (LSTM) layers with 128 units, two dense layers with 2048 units. In early experiments, MAE subjectively turns out to be a better indicator for performance as Mean Squared Error (MSE). Further choosing L1 loss over L2 results in better MAE score. This network trained with no regularization. We produced some predictions using this model, and stored it to a file. Then we started a new simulation in Unity, with the Groundtruth repeating the recorded motion, and the Follower moved by the predictions from the file. Indeed, the produced actuations are able to produce the desired motion but only for some timesteps. As small errors accumulate, the bones drift away, which causes the motion to break. Choosing a good MSE score over a good MAE score results the motion to break faster. This is why MAE was chosen as performance measure for the following analysis.

5.3. Performance Analysis

While testing multiple configurations and settings of hyperparameter to find a network which performs best, we point out some comparison experiments in the next section. The performance analysis is based on multiple training sessions, documented in the appendix B. In the following sections, we refer to those sessions by their IDs. As performance metric, we use the MAE on the performance dataset. This is data which is not part of the training and test data.

5.3.1. Comparison of data processing methods

We explored, which preprocessing operation is best to boost network performance. From analysis of our data and theoretical background, we expect that standard scaling outperforms MinMax scaling. In general, our observations of our training sessions confirm this. Referring to the training sessions A09 & A10 (B.3), we notice training sessions using MinMax preprocessing do archive low loss on the test dataset with preprocessed test batches. However, on the performance dataset the post processed MAE reaches 6.432 & 4.424 compared to other models which are standard scaled, MinMaxScaling reaches worse performance. Comparing the best performing standard scaled models against feature scaled (section 3.5) models, we notice that feature scaling performs well against standard scaling. With A19 (B.5) scoring 0.386 MAE, B2 (B.7) outperforms it and all other standard scaled models, scoring 0.348 MAE.

No-y Scaling

While experimenting with data scaling, on selective sessions y data was not preprocessed. In the logs, these sessions are marked with the special parameter "NoYScaling". Comparing A3 and A4 (B.2) to A19(B.5), it's observable that models without y scaling are not able to compete with A19. As a result, preprocessing the output does improve performance. The same is observable for models C3 and C4 (B.9), which are AllForOne models. Comparing C3 and C4 with other models, we can observe that there are models, which perform better than these. For instance, C13 (B.11). We conclude that preprocessing y of the datasets improves performance.

5.3.2. Training loss comparison

We also explored which loss function performs better, L1 or L2. We expect that L1 loss performance dominates L2 loss. Indeed, models trained by L2 are not able to score low MAE as models trained with L1. Observing A19 versus A17 (B.5) with 0.386 MAE over 0.700 MAE and B2 versus B4 with 0.348 MAE over 0.605 MAE, we note two examples, which confirm the hypothesis that L1 trained models dominate L2 trained models in MAE performance. All of our experiments confirm this finding.

5.3.3. Using Velocity

We created a dataset which includes not only positional and rotational error as input for each bone, but also its current velocity and angular velocity (both $\mathbb{R}^3$). From this dataset 62MV, we expected that models trained by 62MV outperform models trained by 62M. A14 (B.4) showed great promise, scoring well in its test loss, but scores a low MAE of 1.715 on the performance set. All experiments with velocity values added to the dataset caused increased MAE. Here is important to note, that the datasets impose a function from positional and rotational distances to velocities and angular velocities. The mapping is independent from the objects velocity. This is how it can be explained, why the network does not improve with velocity input.

5.3.4. 62M versus 257M

When it comes to neural network training, it is often stated [HNP09] that increasing the amount of data can help produce better results. Therefore, we recorded a bigger dataset (257M) with more samples and compared performance. Contrary to the statement that more data boosts performance, models trained by 257M (B.13) fail to perform as well as A19(B.5). After comprehensive training sessions, we observed the dataset and found that for some entries, the recorder failed to write correct data to the file.

Therefore, we created another dataset 257MR. Comparing sessions trained by 257M (B.13) to 257MR(B.17), the corrected dataset performs with models E1, E2 and E4 better than any recorded result of 257M. Still. A19 (B.5) outperforms all models, which have been trained by 257M or 257MR. At least in the subset of trained networks, the 62M dataset outperforms 257M. So adding more data to the training does not improve the network learning.

5.3.5. Using Collisions

For this experiment, the Collider Dome has been implemented, which we described in section 4.6. A new dataset 62MC has been recorded. This dataset includes collisions, such that boxes push bone segments temporally away. In this experiment, it is tested if added variance can improve performance. With A2 (B.2) and A7-A9 (B.2 B.3), there are recordings of networks trained by 62MC. None of these networks come close to networks trained by 62M.

5.3.6. ActNet type comparison

In section 3.4.1, we discussed how we created different variations of the ActNet. With extensive testing, we compared how these architectures perform against each other. It is to be expected that recurrent layer plus dense layer perform the best. An interesting finding is session A12 (B.3), which scores a MAE of 0.402 . which outperforms most of the other configurations. Especially remarkable is that this network doesn't use recurrent layer and the amount of units is low. Only 193 units perform better than much bigger RNN+DNN models. But no other configuration with only dense layer performed better. This network uses the sigmoid activation function for its dense layers. Nevertheless comparing all sessions against each other, the RNN+DNN approach seems to perform the best. Rather ineffective are convolutional layer before the recurrent ones. Trained with the 257M dataset D10, D11 and D12 B.15 fail to score as well as other models trained by the same dataset. Hence, experimentation with convolutions turned out ot be less effective.

5.3.7. ActNet AllForOne

Section 3.4.2 describes the AllForOne approach in which the data of each bone is used to train a smaller network. Multiple training sessions (B.9) have explored if this more compact network is able to produce better results than one monolithic network. Indeed, some of those sessions, for instance C8, C9 (B.10) and others, produce networks that predict the performance dataset better than regular ActNet modules. From this MAE score, we expect that this network performs better than the regular ActNet. The best model is fully evaluated in the next chapter.

6. Performance Evaluation

Through our experimentation with multiple neural network architectures, we present the performance of this approach.

6.1. ActNet Performance

From all our ActNet training sessions, the best performing architecture emerged from session B02 (B.7) with featured scaling. It's trained by the 62M dataset with l1 kernel regularization and l2 activation regularization with a rate of 0.01. Note that we needed to remove those regularizer from the model json, so that the server is able to load them (see 4.7). This network was trained with a learning rate of 0.001 and Adam optimizer using L1 loss. The dense layer uses Relu activation. The processed sequence length comprises five timesteps. The amount of LSTM units in the first layer are 512, followed by a dense layer of 2048 units. The output layer has 96 units with linear activation. Experiments with more units did not improve the result.

6.1.1. Objective Evaluation

As training session B2(B.7) documents, this network outperforms all other ActNet configurations tested. With a MAE of 0.3475, this network is able to predict the performance dataset better than other architectures tested. Figure 6.1 shows the training and test loss during the training session.

In figure 6.1, training and test loss are plotted against the number of training iterations. These values are much lower than the MAE on the performance dataset, because this loss is calculated on the preprocessed data, whereas the performance metric is post processed. We observe that the test loss converges fast, while the training loss continues to decrease. Especially towards the last iterations, the training loss is about to decrease further, whereas the test loss continues to maintain its level. The best value for the test loss has been recorded at 2995 iterations, therefore, an early stopping at 2995 iterations is appropriate to prevent overfitting. From the graph it is observable that further training does not improve performance. Compared to session B6 and B7 (B.8) the regularization techniques used in B2 improved the result.

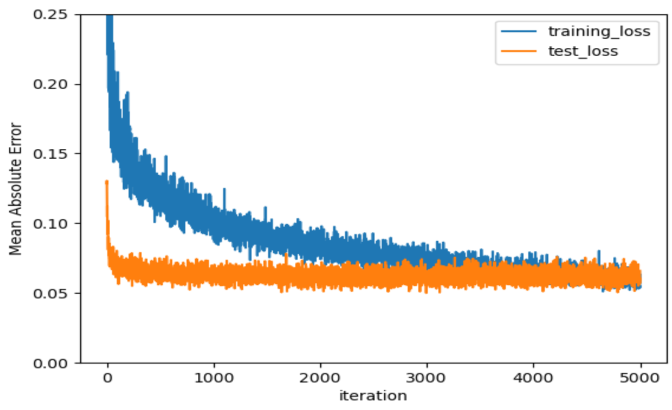


Figure 6.1.: B2 loss during training

6.1.2. Subjective Evaluation

For a subjective performance analysis, we load this model on the Ubi server and run Unity to observe it. The Groundtruth avatar performs motions via animator, the Follower publishes its state to the Ubi Server and this model responds with its predicted actuators. First, to observe the absolute performance of the network's predictions, we use Unity's step feature, which allows us to execute one simulation step on demand. Therefore, the client server structure has enough time to communicate for each timestep predictions. This ensures that potential synchronization problems or lag are not affecting the evaluation.

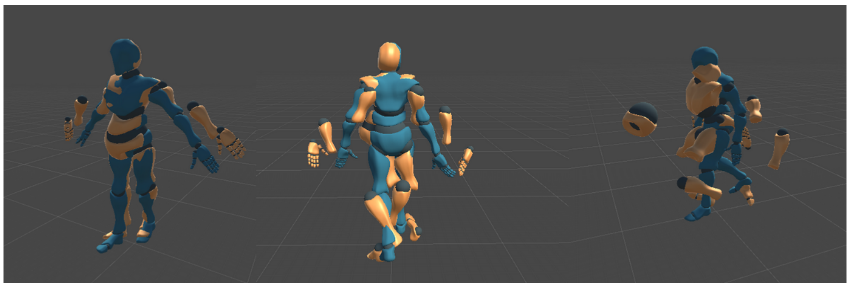


Figure 6.2.: From left to right three captures of B2 predicting a motion imposed by blue.

In figure 6.2 we see three screenshots of the avatar. The blue avatar is the animated one, while the orange one is actuated by the network. Even though the network has to await 5 published timesteps to make its first actuations, in the first capture to the left of figure 6.2, the network can follow the other avatar to some extent. The legs are in place, the arms are rotated correct, but the positions drift already away. In the second capture, it's observable how stability of the avatar starts to break. Segments are not connected anymore, the lower arms and hands were not able to correct their early displacement enough. Head, chest and hip bones such as upper arms seem similar in position and rotation. In the last capture to the far right we can see that the avatar broke apart. The network is not able to keep the segments together or to mimic the motion.

6.1.3. Comparison to Model A19

Since, B2 is not able to produce correct motion, we compare it to the model trained by A19. It is trained by the 62M dataset with the regular row-wise standard scaling. It has no regularizations, 2048 lstm units and 4096 dense units. The output layer has 96 units with linear activation. With a MAE score of 0.3680, it is the best performing model which is not feature scaled.

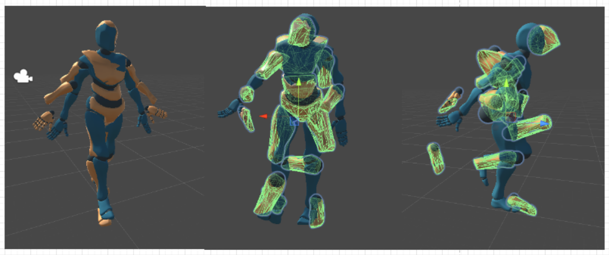


Figure 6.3.: From left to right three captures of A19 predicting a motion imposed by blue.

Again, in figure 6.3 three captures were recorded in one simulation. At the beginning, the segments rotate into their correct desired rotation, the Follower's orange arms translate downwards like the Groundtruth ones. In the second capture, the network's predictions strafe away from the desired motion. Some segments have incorrect positions and rotations, while others are still close. Comparing the middle capture of figure 6.3 to figure 6.2, it seems that this model fails to rotate correctly, while the other one fails to translate correctly. In the last capture, we see that the predictions can't correct the errors which have accumulated already. Stability is broken. In total, our best trained ActNets are not able to generalize well. Since both models, A19 and B2, use different pre and post processing techniques, an implementation error seems to be unlikely.

6.2. ActNet AllForOne Performance

To evaluate the performance of ActNet AllForOne, we choose the model created by session C9 (B.10). Objectively, it doesn't have the smallest MAE, unlike the model created in session C8 (B.10), but it has been built without regularizer and the margin in performance is very low. To prevent the issue in section 4.7, in which models with keras regularizer can't be loaded on the server to affect our evaluation, we choose C9 over C8. This network takes 6 input values, comprising the six input features of each bone. It is constructed with one LSTM layer and 1024 units, with sigmoid activation. It is followed by two 1024 units of dense layers activated by Relu. Its final layer is a dense layer with 6 values and linear activation. It has been trained with the Adam Optimizer, with 10000 iterations and L1 loss on the 62M dataset. Networks with more units did not improve the result.

6.2.1. Objective Evaluation

In its performance, the network trained in session C9 B.10, scores a MAE of 0.2858 which is lower than the MAE scores of any regular ActNet models.

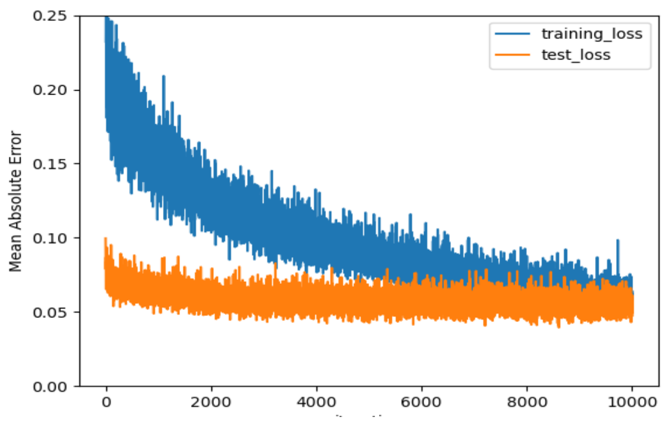


Figure 6.4.: C9 loss during training

Observing the loss plot in figure 6.4, the test loss has converged quickly. They best test loss has been observed in iteration 9950. So there might be potential to increase performance with more training.

Let's further evaluate the MAE with UNSCALED MAE meaning the postscaled metric:

```
----------Results-----------
Total L2 Test Loss: 0.28794831113008956
Total POSTSCALED MAE: 0.28580278398667397
Total POSTSCALED MAE Features: [0.07473 0.05835 0.08585 0.51198 0.46792 0.51598]
```

Above, we see metrics on the performance dataset of the model trained in C9. The first three values are the loss for predicted velocity in x,y and z and the last three values for angular velocity in x,y and z. For the MAE, we can observe that on average the absolute error in predicted velocity is smaller than predicted angular velocity. Let's not forget that the output of this network are first order velocities, which are passed to AddForce() and AddTorque() after the values are integrated over time. Therefore, we can expect an average displacement in position. With an example MAE velocity of 0.07 and a time step of 0.02 delta time, this makes an average displacement of around 0.0014 displacement units. With the avatar having a size of roughly 1.75 units, we can conclude that the error drift is low. Nevertheless, one worse case prediction can destroy the stability of the avatar.

6.2.2. Subjective Evaluation

Loading this model into Unity, it is observable that the predictions of this model do not seem to recreate the desired motion at all.

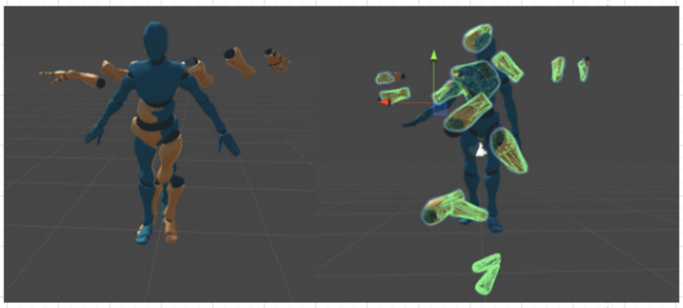


Figure 6.5.: Subjectively, this network performs worse than the previous ActNet.

As visible in figure 6.5, this network does not recreate the motion even after the first computational steps with predictions. The arm segments don't translate. On the left of figure 6.5, we can observe a screenshot after some initial steps. The arm segments fail to translate downwards, while the rotations on the right side seem to be correct. Segments with little translation seem to be moving fine. After some more steps, it is clear that the predictions are not able to capture the desired motion. During play, the segments keep their wrong positions and rotate continuously around a fixed spot. Compared to the subjective evaluation of A19 and B2, this network performs less in the Unity environment. Why is the subjective performance of the AllForOne approach so much worse? This is answered by observing the time the server needs to process the predictions. The AllForOne approach requires the smaller network to run fully for each bone. Which means, the network runs sixteen times. Even if we step through the simulation slowly, the client does not get new data from the server, therefore the rigidbodies maintain their velocities and their angular velocities. This explains the continuous spinning we observed earlier. The client server implementation seems to be not feasible here and reaches its limits, or there are synchronization problems, which are not accounted for.

7. Conclusion

Let's come to a conclusion. The performance evaluation yields that even the best models lose stability. This suggests that the network is not able to counter react its rigidbodies drifting away from each other. Increasing the recurrent networks sequence length or increasing the amount of units did not improve the result. We also observed that the current client server process is not synchronized well enough to communicate in runtime. Other experiments such as adding the velocities of the rigidbodies to the networks input or recording the dataset with forced collisions did not improve the performance. We come to the conclusion that this approach can not create the required stability for motion controlling a humanoid avatar. In section 6.2.1, we observed that the displacement on predicted values in the performance is low. This suggest that a good performance on training data and performance dataset is not able to generalize well enough in the running simulation.

7.1. Further work

In this approach, the avatar is not able to maintain stability. Bones drift away from each other. At places where joints would be, segments disconnect. The networks actuators do not learn to maintain stability. To enforce it, the neural network loss could be further constrained by adding a penalty [Set97] to the loss function, once adjacent segments break or the distance of bones to their joints increase. This could force the network to learn a solution that keeps the segments together. While searching for the optimal hyperparameters, this work only covers a small subset of potential hyperparameters, in that regard there are still more configurations to be explored. While we observe, how the Rigidbodies disconnect we notice that the segments are not rotated around their joints. It could be explored to setup a more constrained avatar, which is only rotated around its joints with limited DoF. While this work tries to actuate the avatar by a neural network only, it could also be explored to develop motion control with different controllers which can be switched, like Chen and Narendra [CN01] propose. Based on their idea, the avatar could be moved arithmetical if there are no collisions expected. Whereas, once collisions happen, or other cases in which solving actuations mathematically does not suffice, the controller is switched to a neural network which is trained for these special cases. The current client server implementation faces the issue that the process of sending data to the server, making predictions there and posting them back to Unity creates synchronisation problems. This can be observed as soon as we choose to go into uninterrupted play mode in Unity. The client server communication can not keep up with the physical calculations rate of 0.02 seconds. This problematic is even more present during the AllForOne approach (see section 6.2.2). Therefore, there is more implementation necessary for synchronizing Tensorflow predictions to Unity at runtime.

A. Data Histograms

The following pages show plots of histograms of some features of the data sets used in this work.

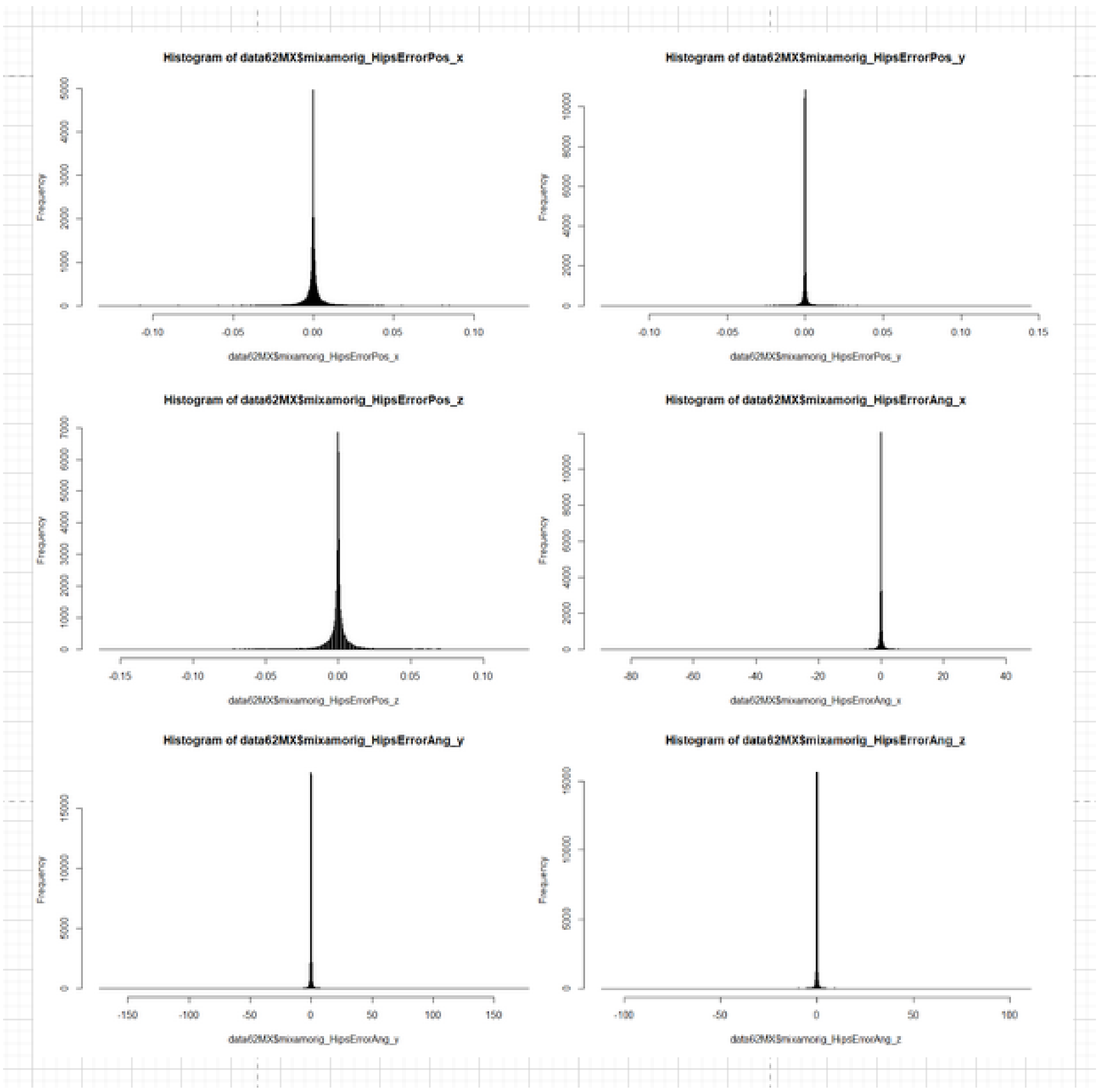


Figure A.1.: 62M_Hip_X

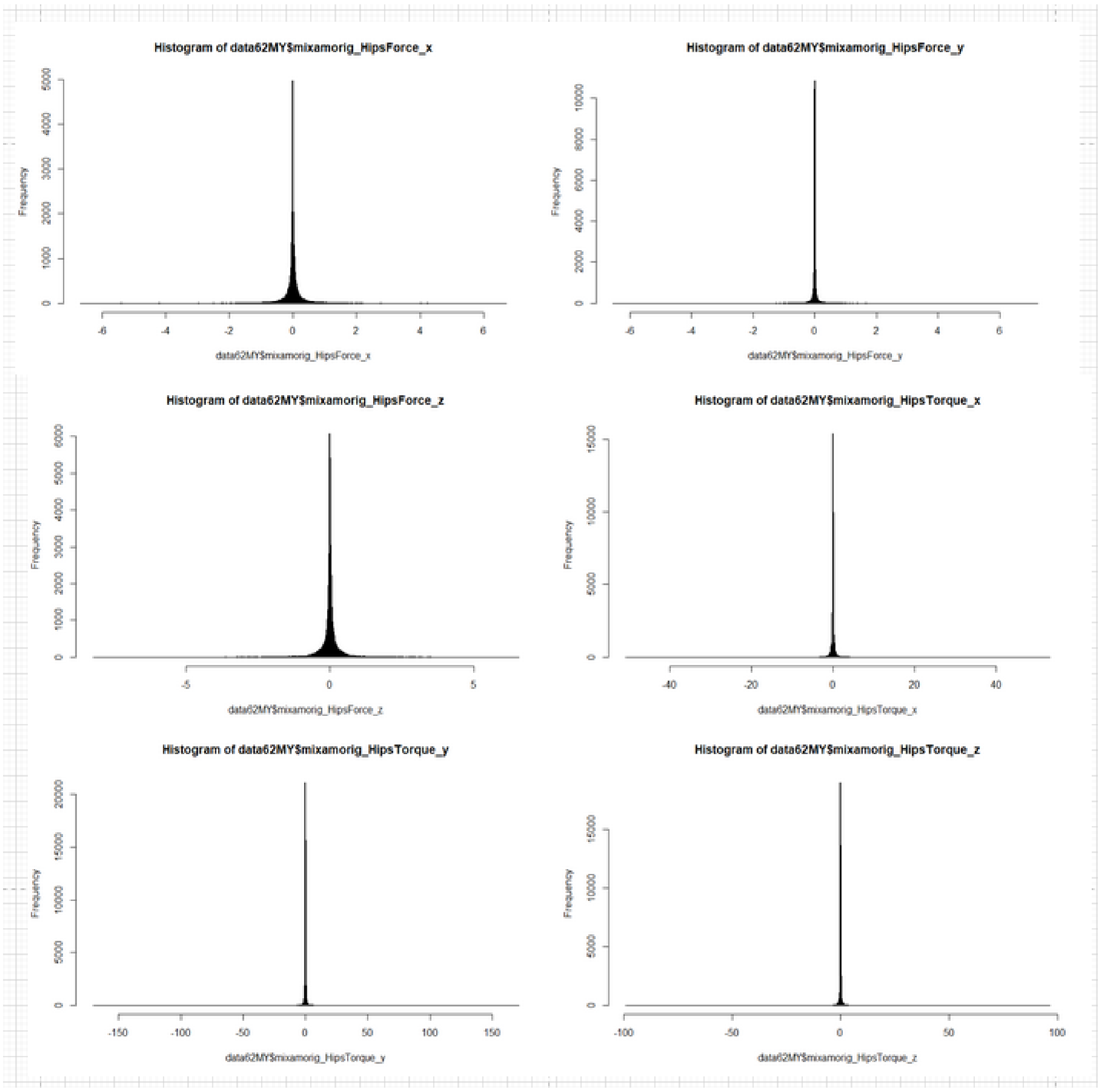


Figure A.2.: 62M_Hip_y

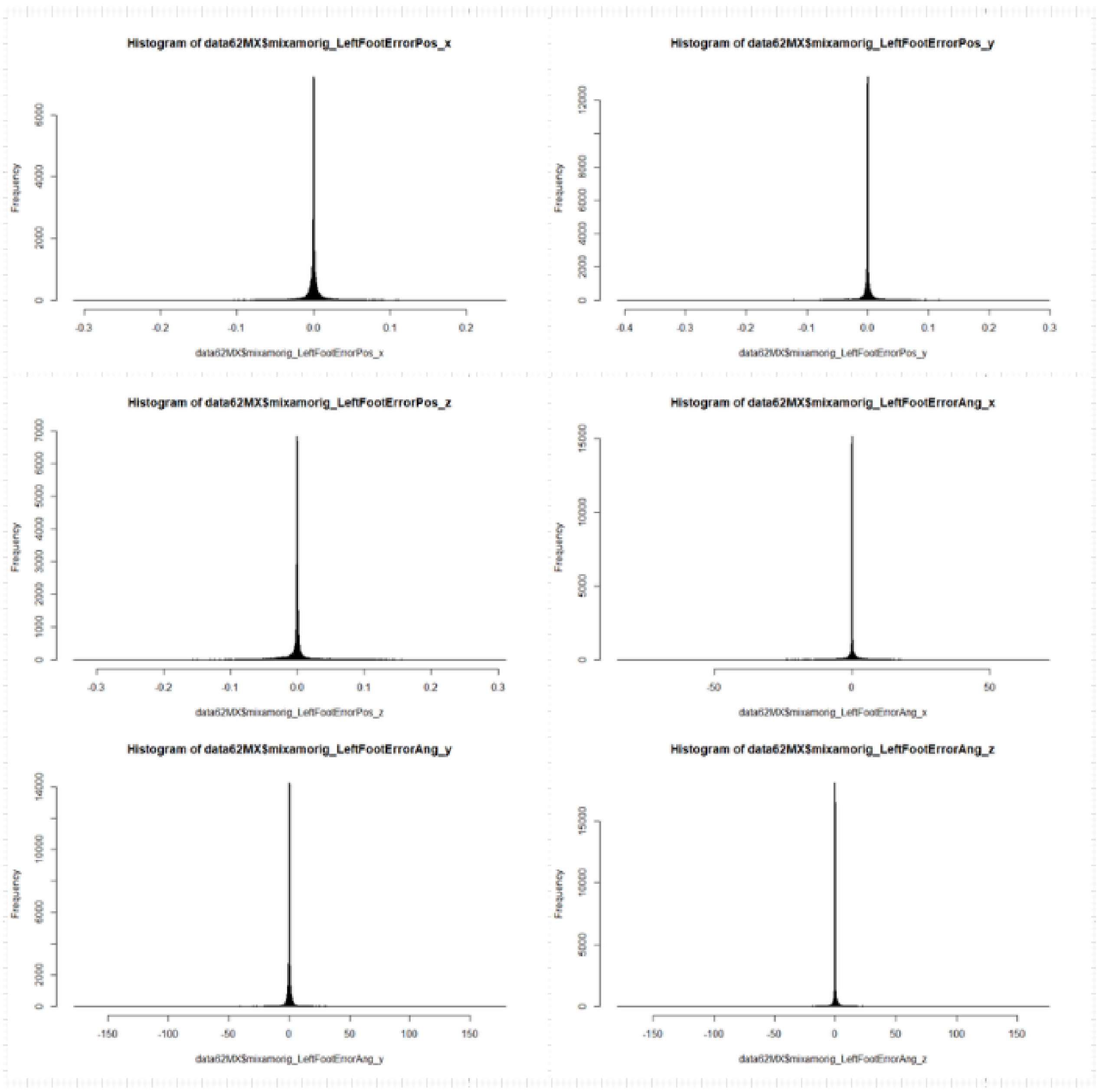


Figure A.3.: 62M_LeftFoot_X

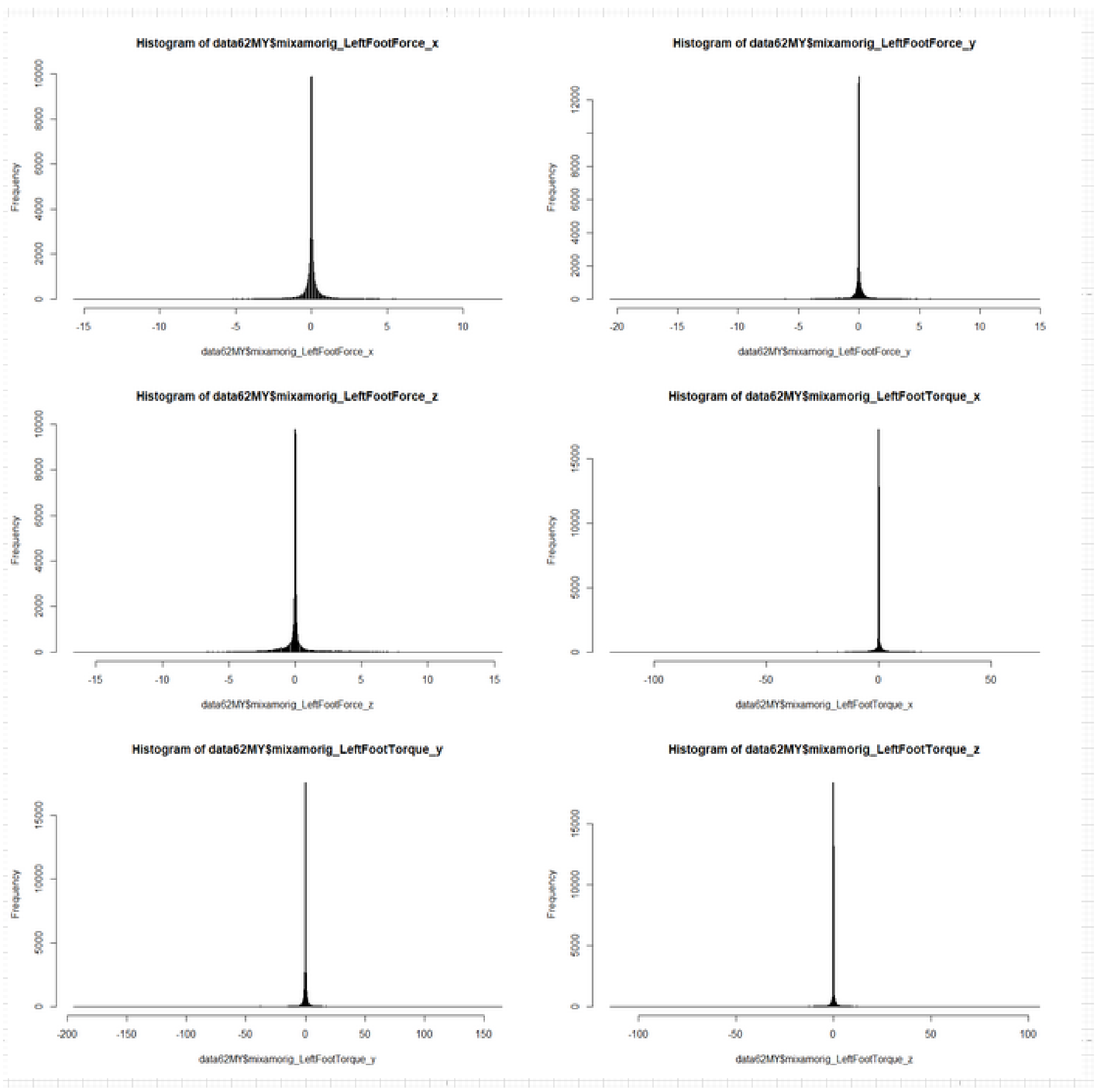


Figure A.4.: 62M_LeftFoot_y

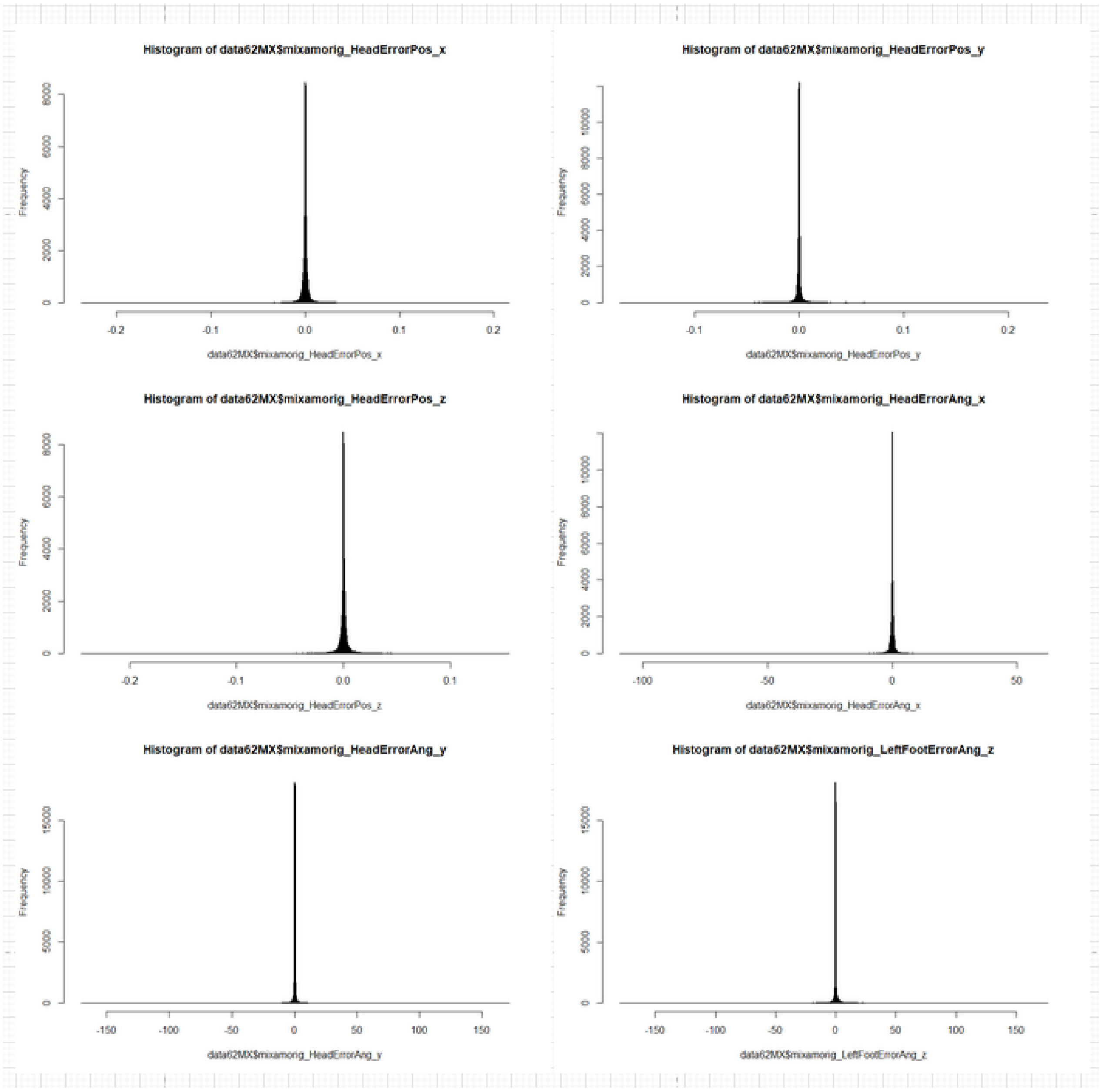


Figure A.5.: 62M_Head_X

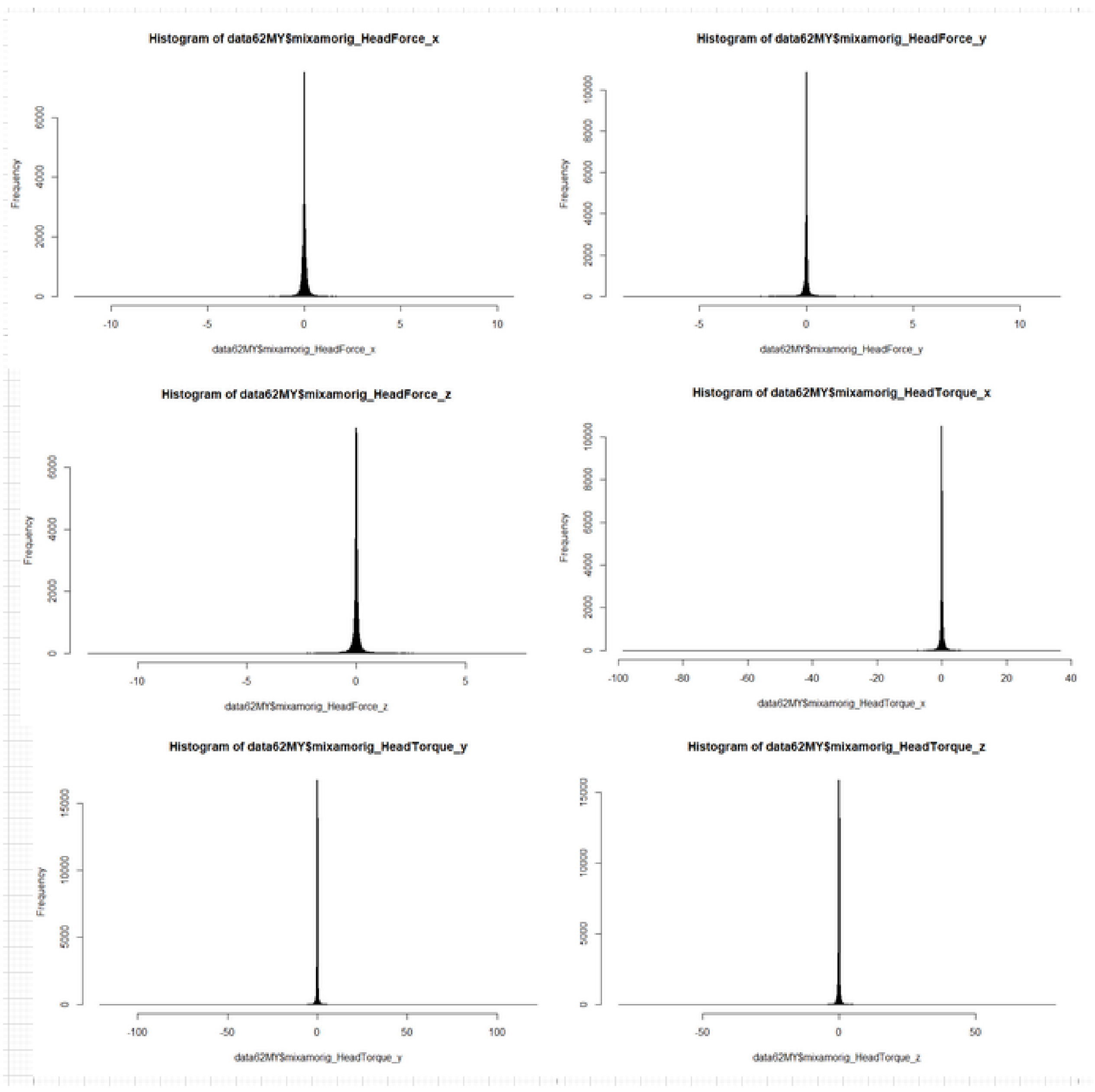


Figure A.6.: 62M_Head_y

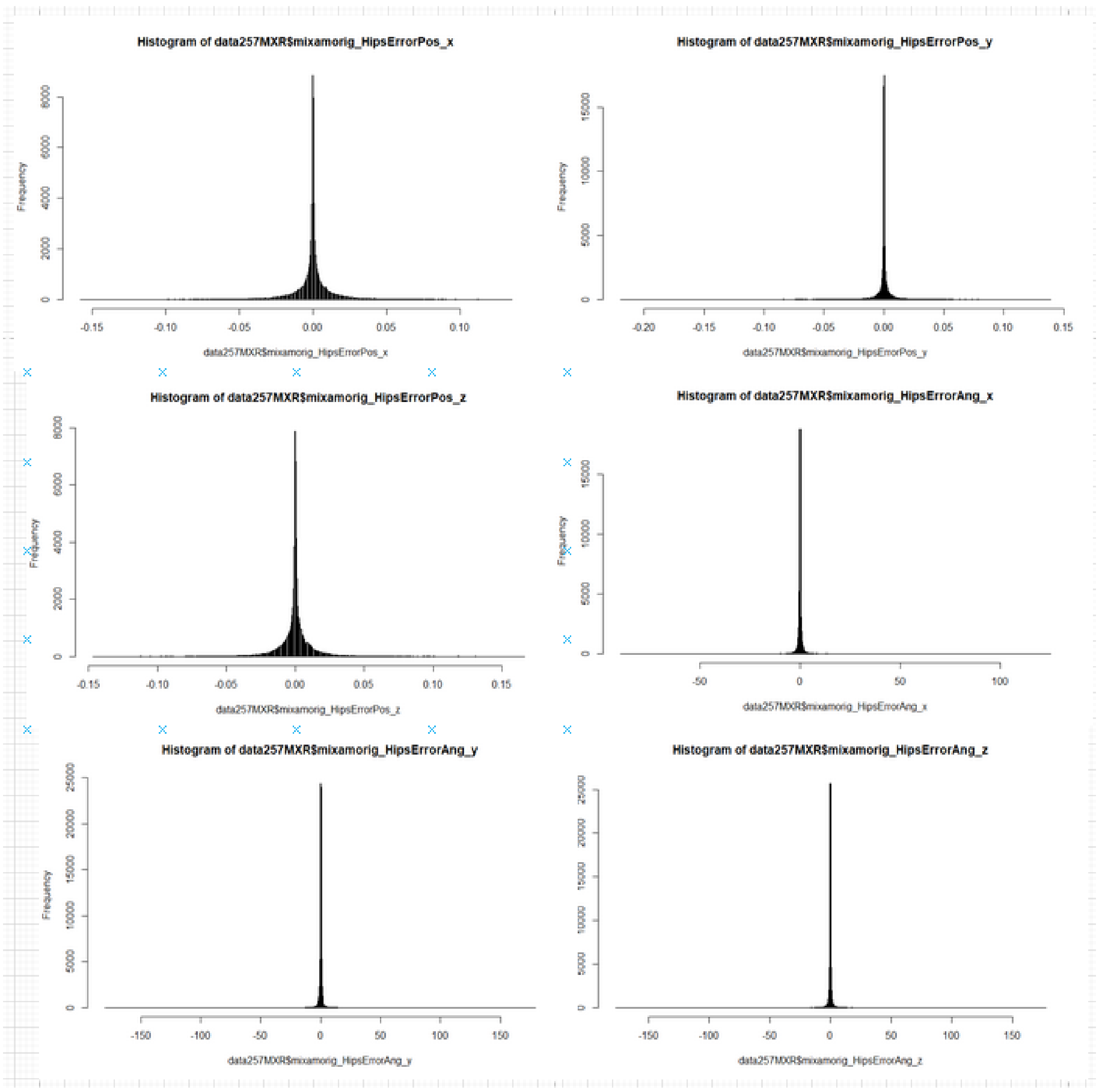


Figure A.7.: 257MR_Hip_X

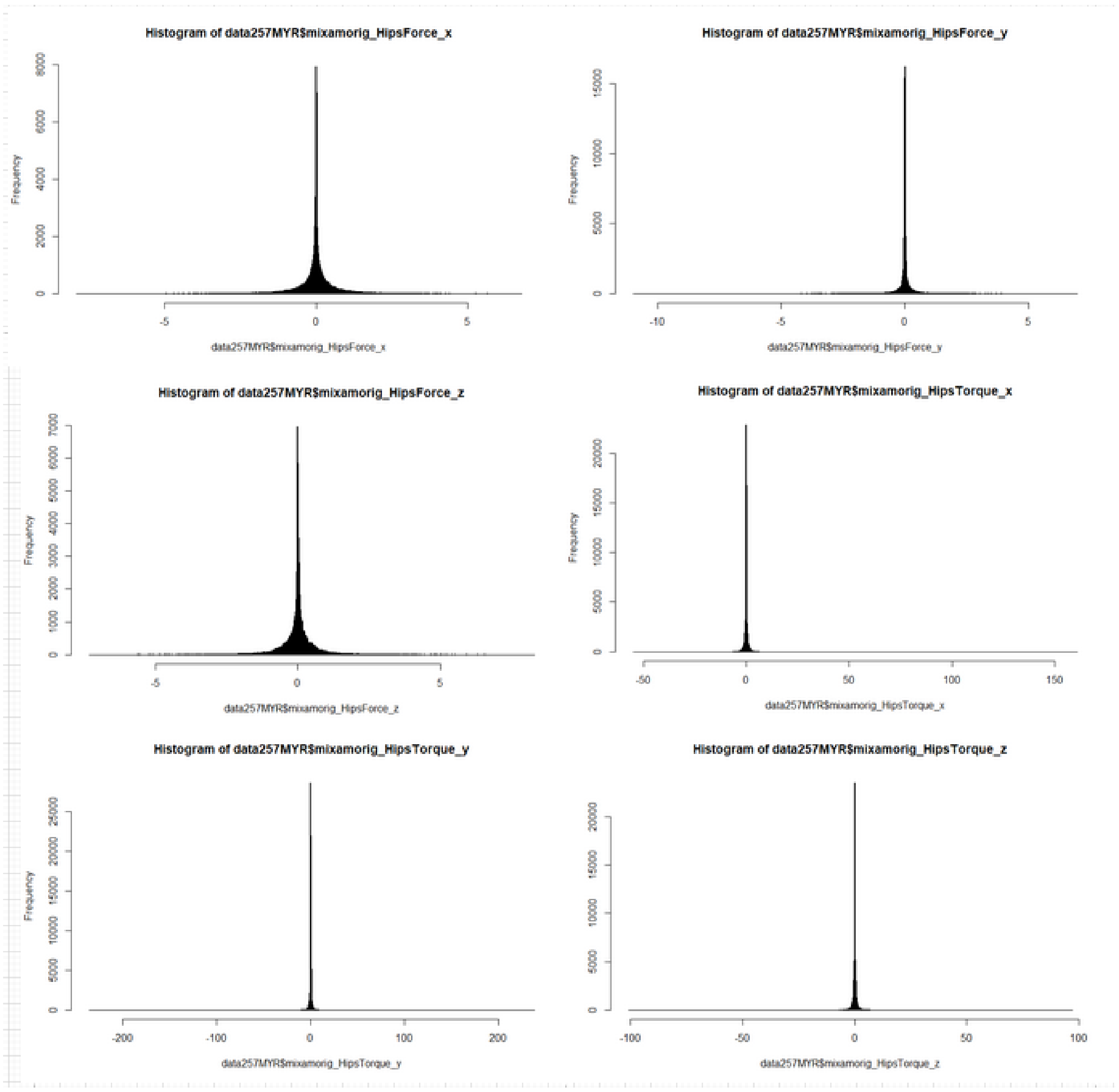


Figure A.8.: 257MR_Hip_y

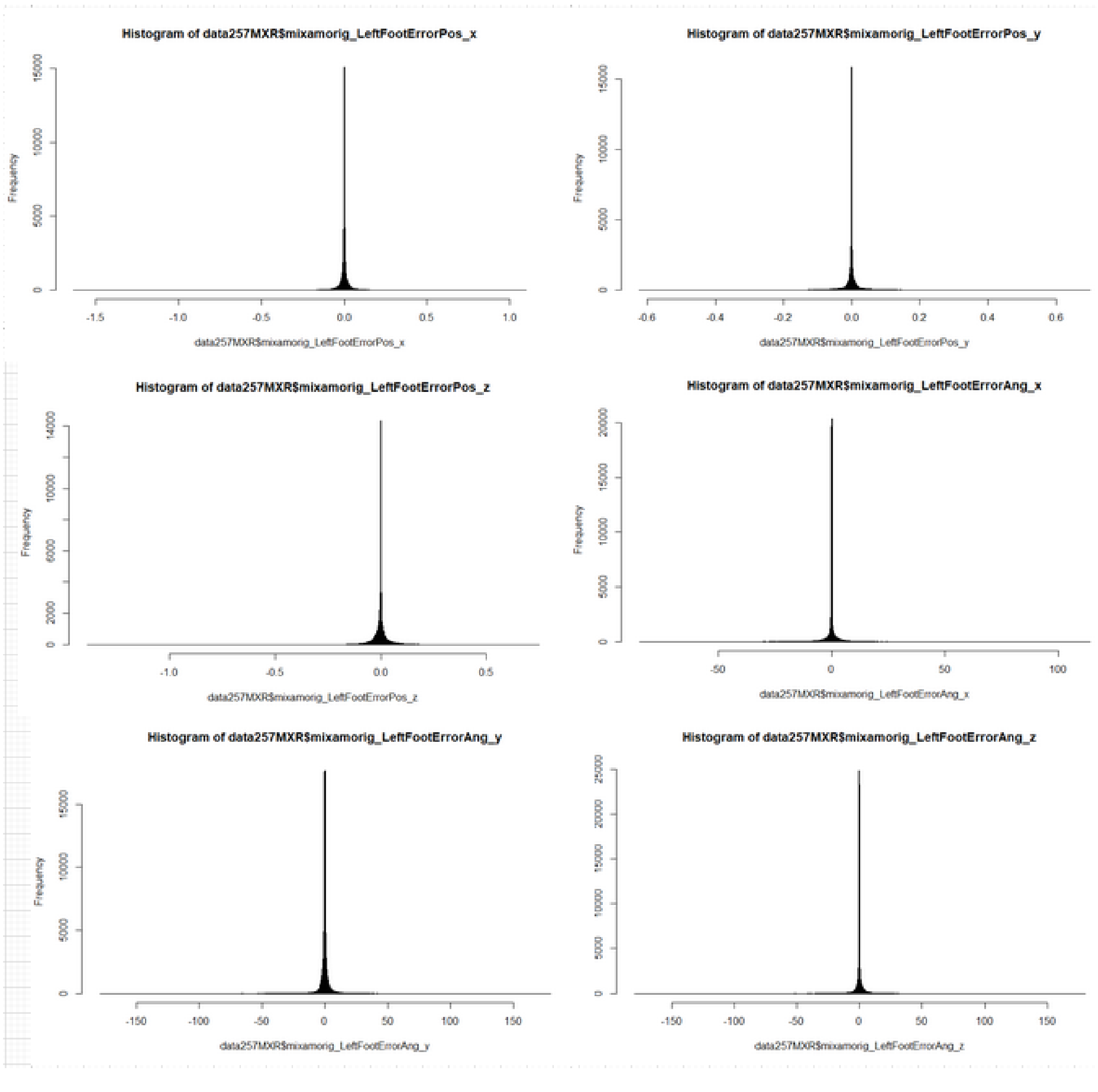


Figure A.9.: 257MR_LeftFoot_X

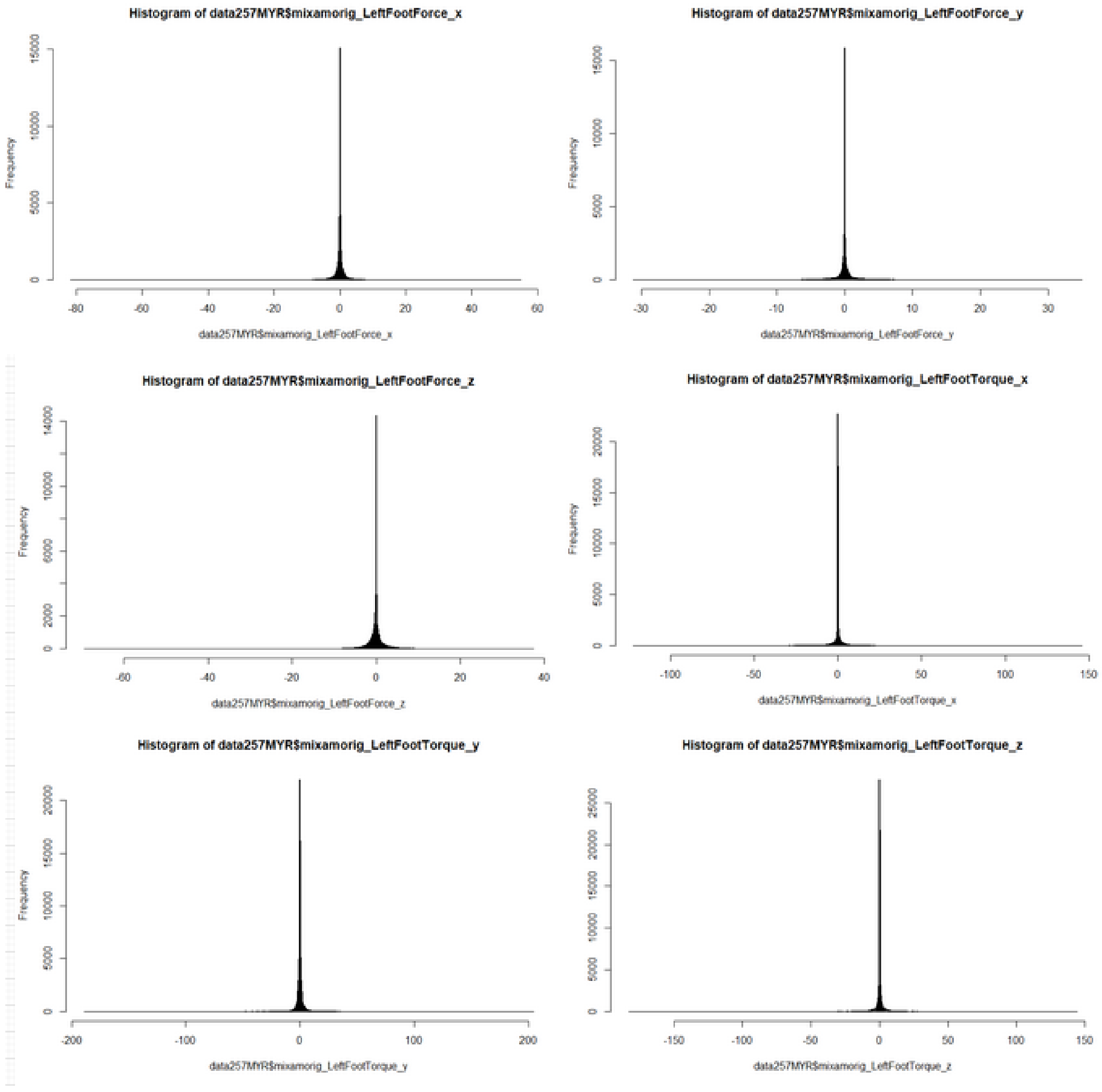


Figure A.10.: 257MR_LeftFoot_y

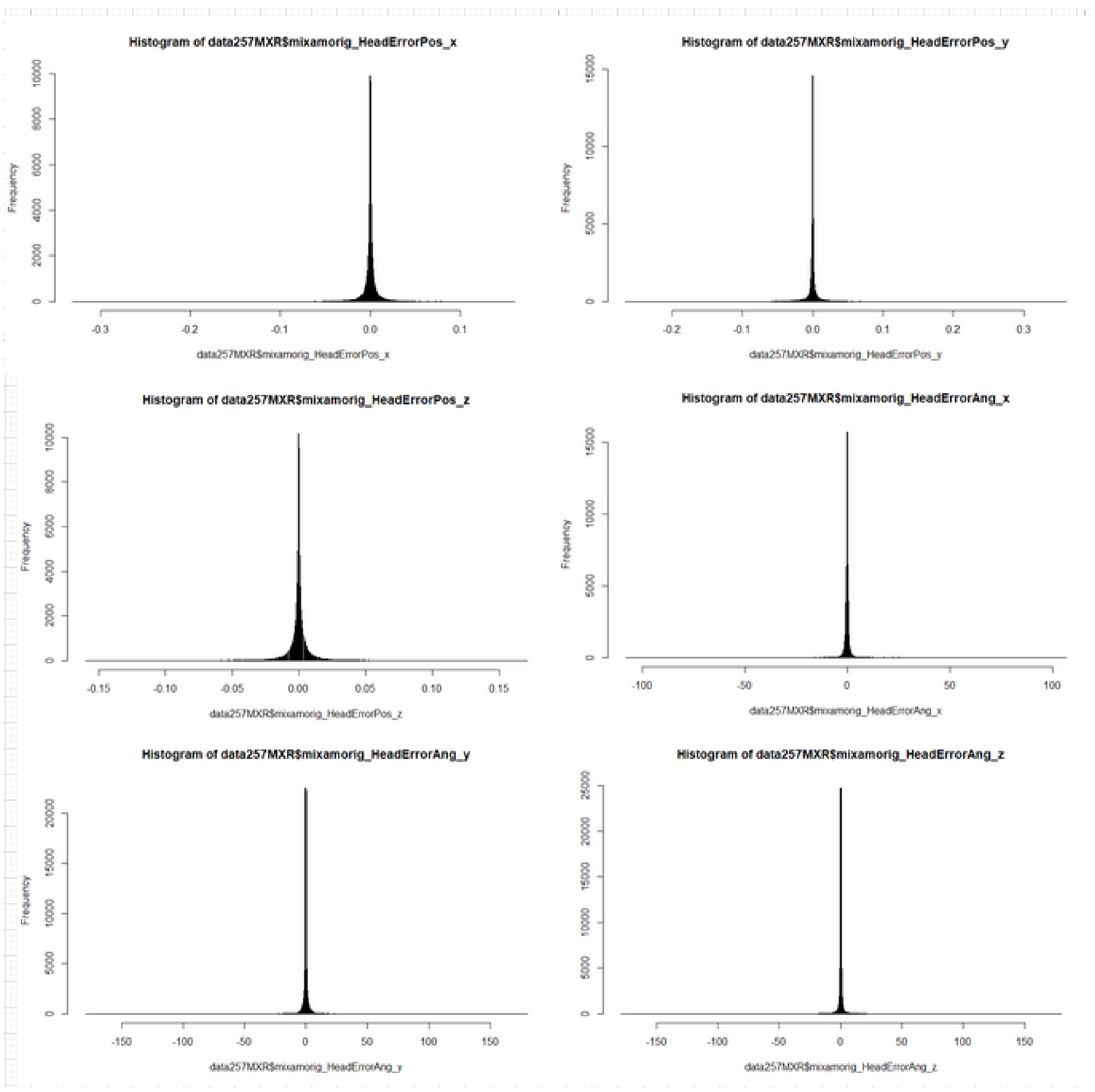


Figure A.11.: 257MR_Head_X

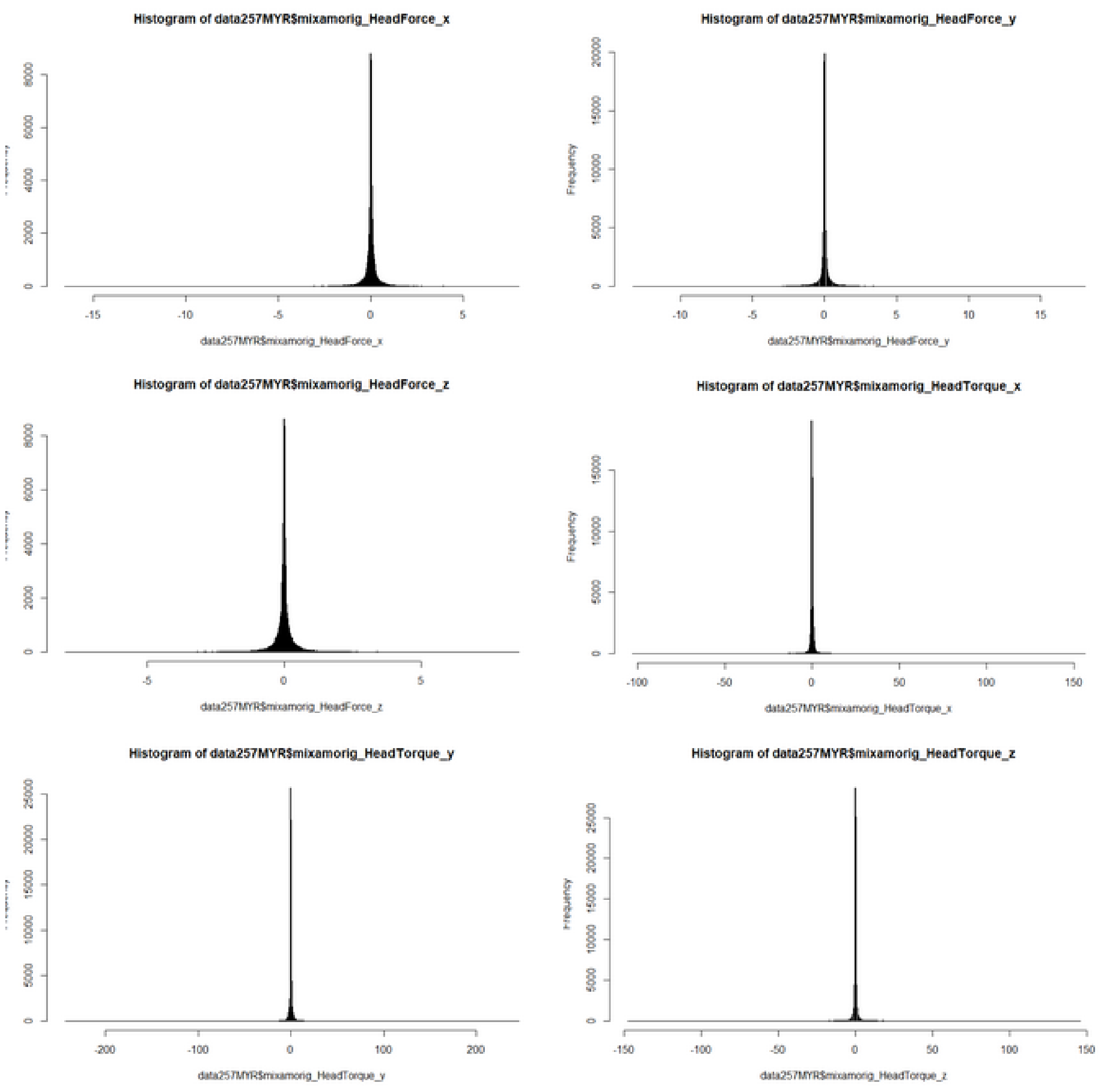


Figure A.12.: 257MR_Head_y

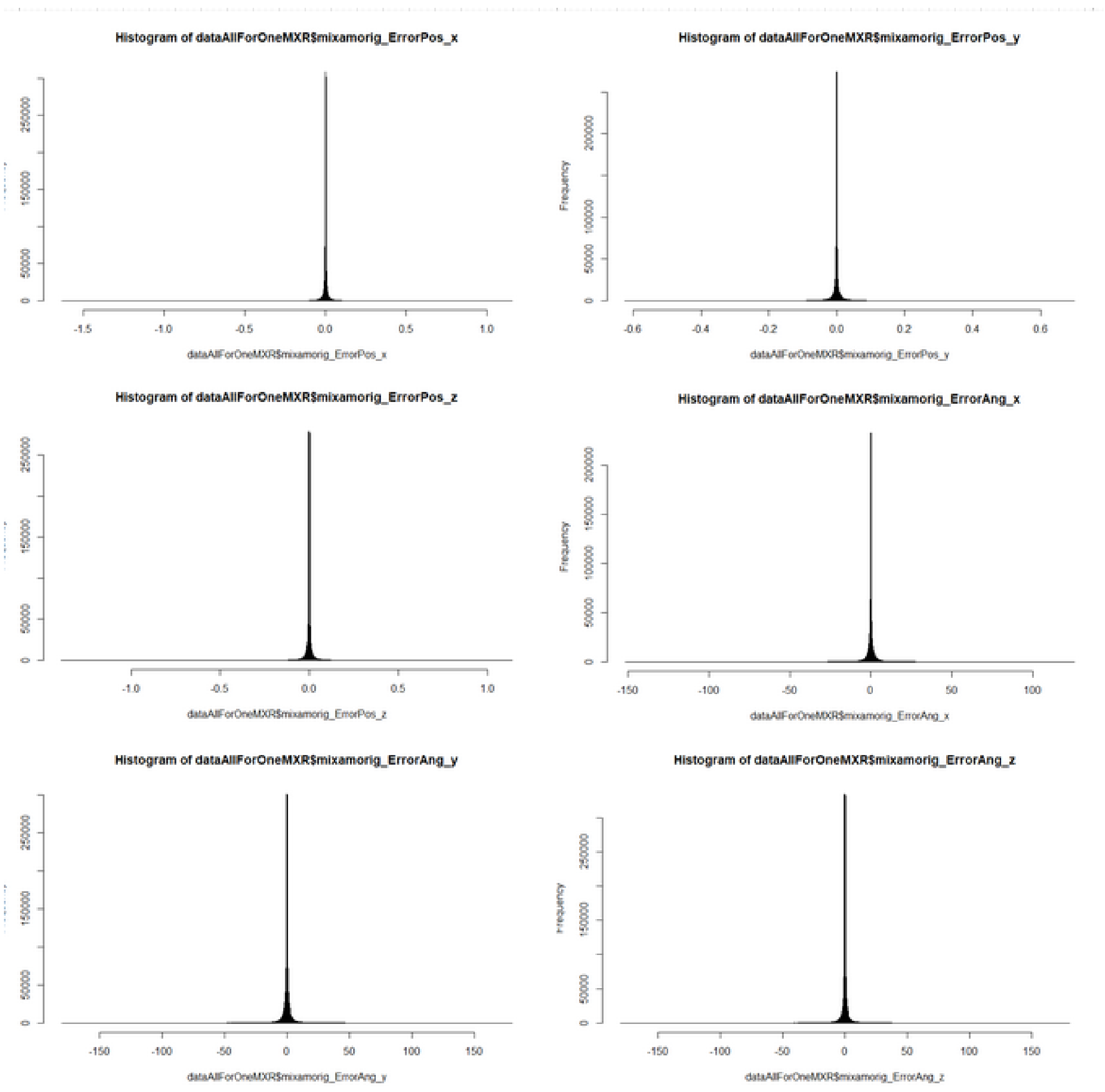


Figure A.13.: 257MR_AllForOne_X

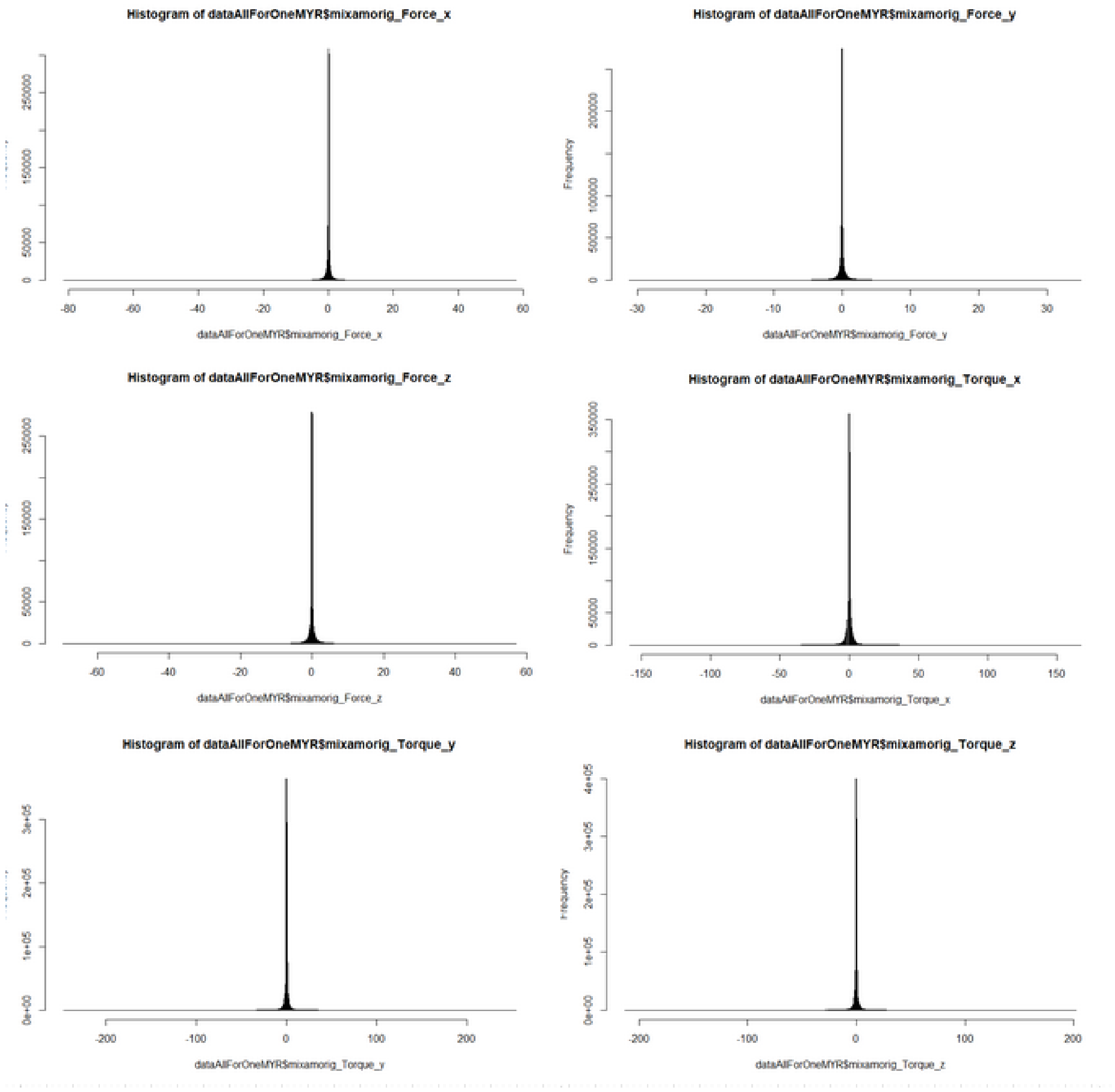
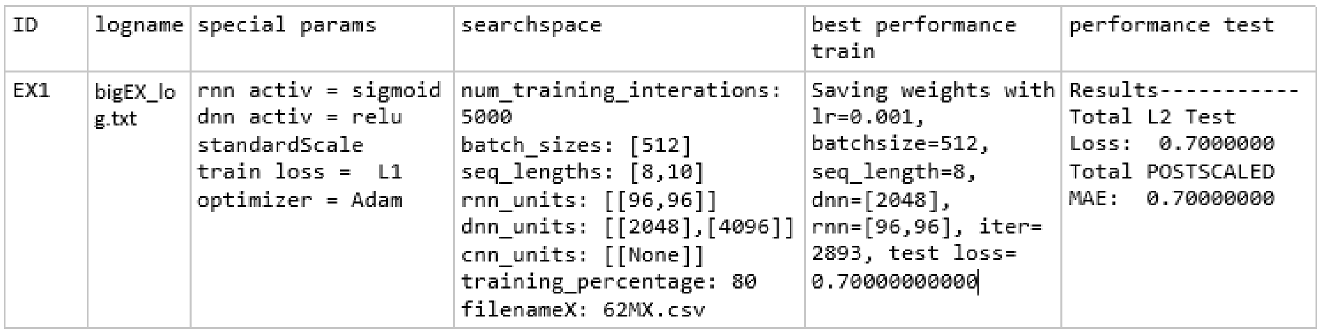


Figure A.14.: 257MR_AllForOne_y

B. Training logs

B.1. How to read the logs

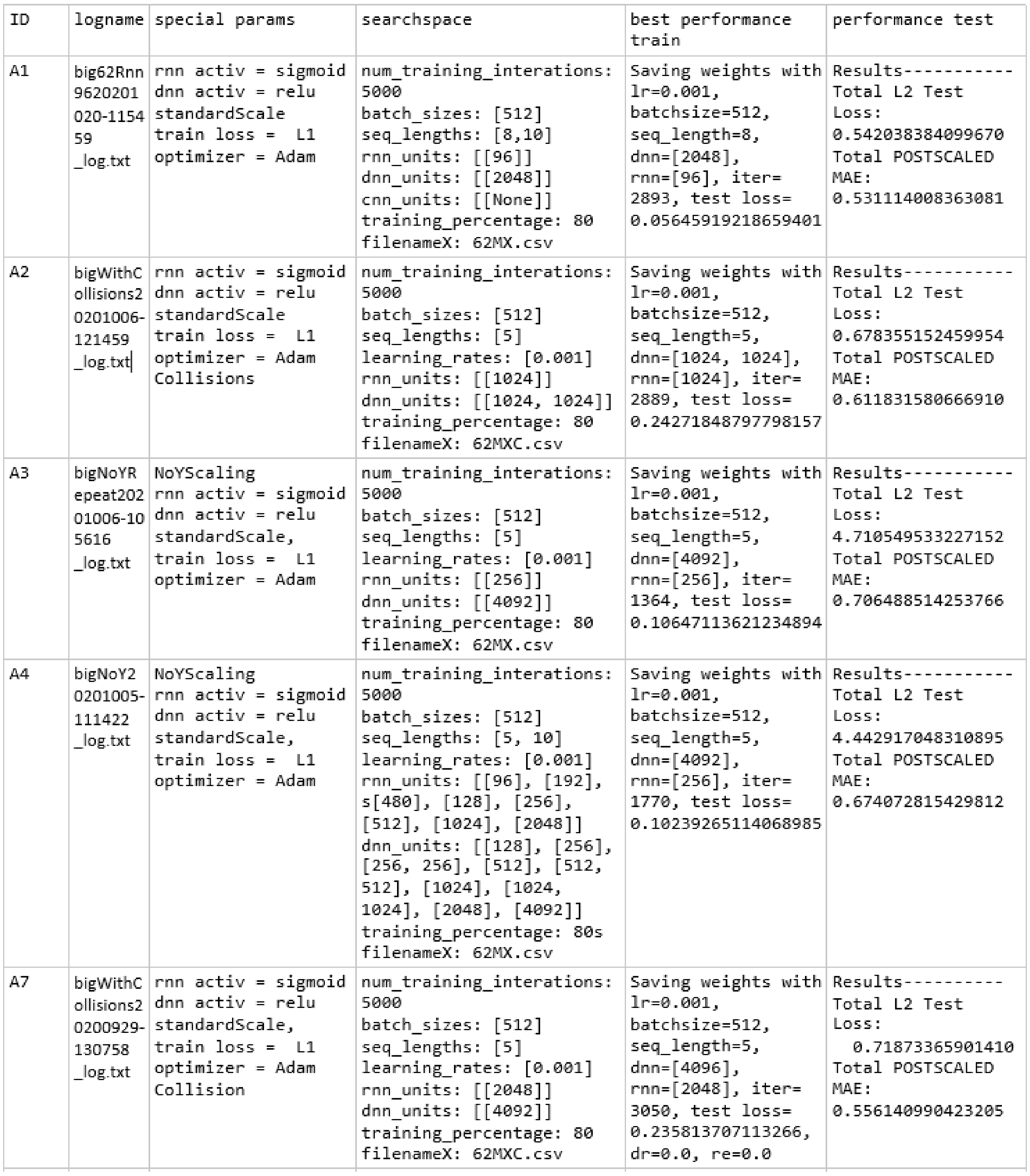
Each training session produces a log as the following ones. Row special params informs about parameters which are the same for all training runs in the session, whereas row searchspace consists of hyperparameters which are iterated over.



ID	logname	special params	searchspace	best performance train	performance test
EX1	bigEX_log.txt	rnn activ = sigmoid dnn activ = relu standardScale train loss = L1 optimizer = Adam	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [8,10] rnn_units: [[96,96]] dnn_units: [[2048],[4096]] cnn_units: [[None]] training_percentage: 80 filenameX: 62MX.csv	Saving weights with lr=0.001, batchsize=512, seq_length=8, dnn=[2048], rnn=[96,96], iter= 2893, test loss= 0.70000000000	Results----------- Total L2 Test Loss: 0.7000000 Total POSTSCALED MAE: 0.70000000

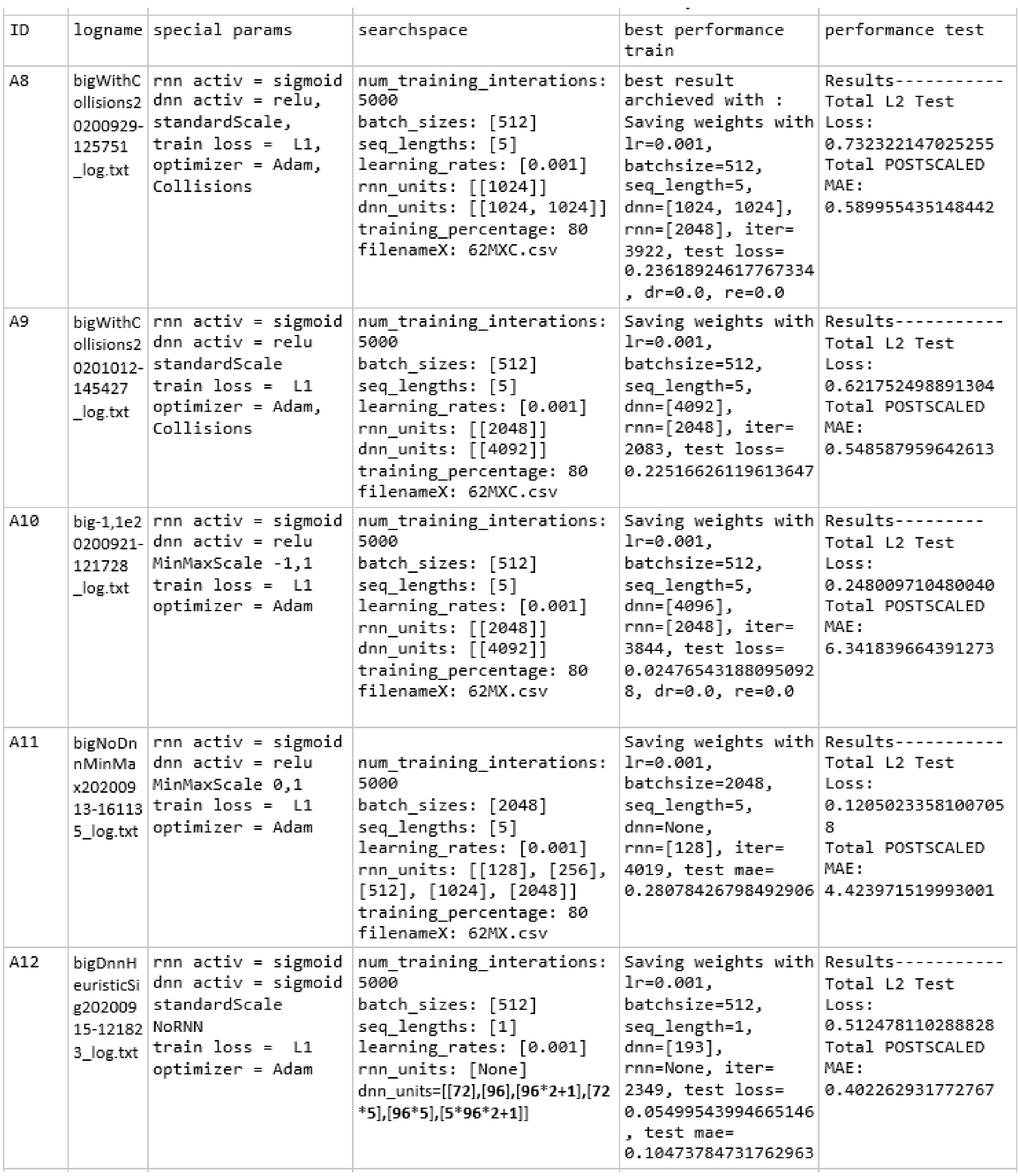
Figure B.1.: LogExample

Figure B.1 shows an example of a log entry. This session with ID EX1 was trained with sigmoid activation on the recurrent layer and Relu activation on dense layer. The data was preprocessed with standard scaling. Adam has been used as optimizer and L1 loss. The searchspace column shows that all models have been trained with 5000 training iterations with a batch size of 512. Sequence lengths of 8 and 10 were tried. Two recurrent layer with 96 units were used. For the dense layer, two settings were tried. Either a dense layer with 2048 units or a dense layer with 4096 layer. No convolutional layers were tested in this sessions. Therefore this log trained 4 different models. In the row best performance train, we see which combination of hyperparameter from the searchspace trained the best model. This is measured by the test loss. Finally in the row performance test, the post scaled MAE on the performance set it presented.



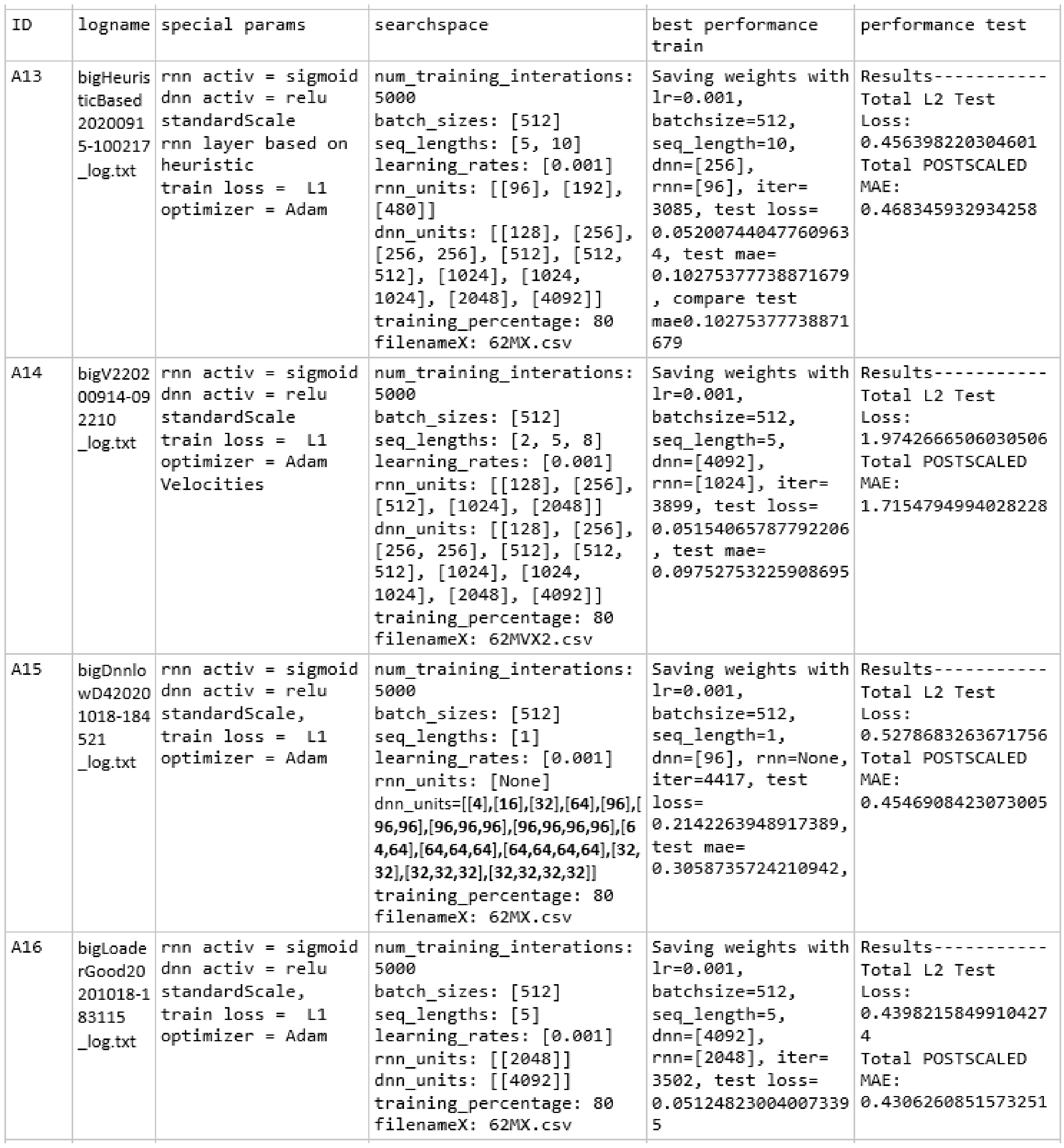
ID	logname	special params	searchspace	best performance train	performance test
A1	big62Rnn 9620201 020-1154 59 _log.txt	rnn activ = sigmoid dnn activ = relu standardScale train loss = L1 optimizer = Adam	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [8,10] rnn_units: [[96]] dnn_units: [[2048]] cnn_units: [[None]] training_percentage: 80 filenameX: 62MX.csv	Saving weights with lr=0.001, batchsize=512, seq_length=8, dnn=[2048], rnn=[96], iter= 2893, test loss= 0.05645919218659401	Results----------- Total L2 Test Loss: 0.542038384099670 Total POSTSCALED MAE: 0.531114008363081
A2	bigWithC ollisions2 0201006- 121459 _log.txt	rnn activ = sigmoid dnn activ = relu standardScale train loss = L1 optimizer = Adam Collisions	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.001] rnn_units: [[1024]] dnn_units: [[1024, 1024]] training_percentage: 80 filenameX: 62MXC.csv	Saving weights with lr=0.001, batchsize=512, seq_length=5, dnn=[1024, 1024], rnn=[1024], iter= 2889, test loss= 0.24271848797798157	Results----------- Total L2 Test Loss: 0.678355152459954 Total POSTSCALED MAE: 0.611831580666910
A3	bigNoYR epeat202 01006-10 5616 _log.txt	NoYScaling rnn activ = sigmoid dnn activ = relu standardScale, train loss = L1 optimizer = Adam	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.001] rnn_units: [[256]] dnn_units: [[4092]] training_percentage: 80 filenameX: 62MX.csv	Saving weights with lr=0.001, batchsize=512, seq_length=5, dnn=[4092], rnn=[256], iter= 1364, test loss= 0.10647113621234894	Results----------- Total L2 Test Loss: 4.710549533227152 Total POSTSCALED MAE: 0.706488514253766
A4	bigNoY2 0201005- 111422 _log.txt	NoYScaling rnn activ = sigmoid dnn activ = relu standardScale, train loss = L1 optimizer = Adam	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [5, 10] learning_rates: [0.001] rnn_units: [[96], [192], s[480], [128], [256], [512], [1024], [2048]] dnn_units: [[128], [256], [256, 256], [512], [512, 512], [1024], [1024, 1024], [2048], [4092]] training_percentage: 80s filenameX: 62MX.csv	Saving weights with lr=0.001, batchsize=512, seq_length=5, dnn=[4092], rnn=[256], iter= 1770, test loss= 0.10239265114068985	Results----------- Total L2 Test Loss: 4.442917048310895 Total POSTSCALED MAE: 0.674072815429812
A7	bigWithC ollisions2 0200929- 130758 _log.txt	rnn activ = sigmoid dnn activ = relu standardScale, train loss = L1 optimizer = Adam Collision	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.001] rnn_units: [[2048]] dnn_units: [[4092]] training_percentage: 80 filenameX: 62MXC.csv	Saving weights with lr=0.001, batchsize=512, seq_length=5, dnn=[4096], rnn=[2048], iter= 3050, test loss= 0.235813707113266, dr=0.0, re=0.0	Results---------- Total L2 Test Loss: 0.71873365901410 Total POSTSCALED MAE: 0.556140990423205

Figure B.2.: LogsA01-07



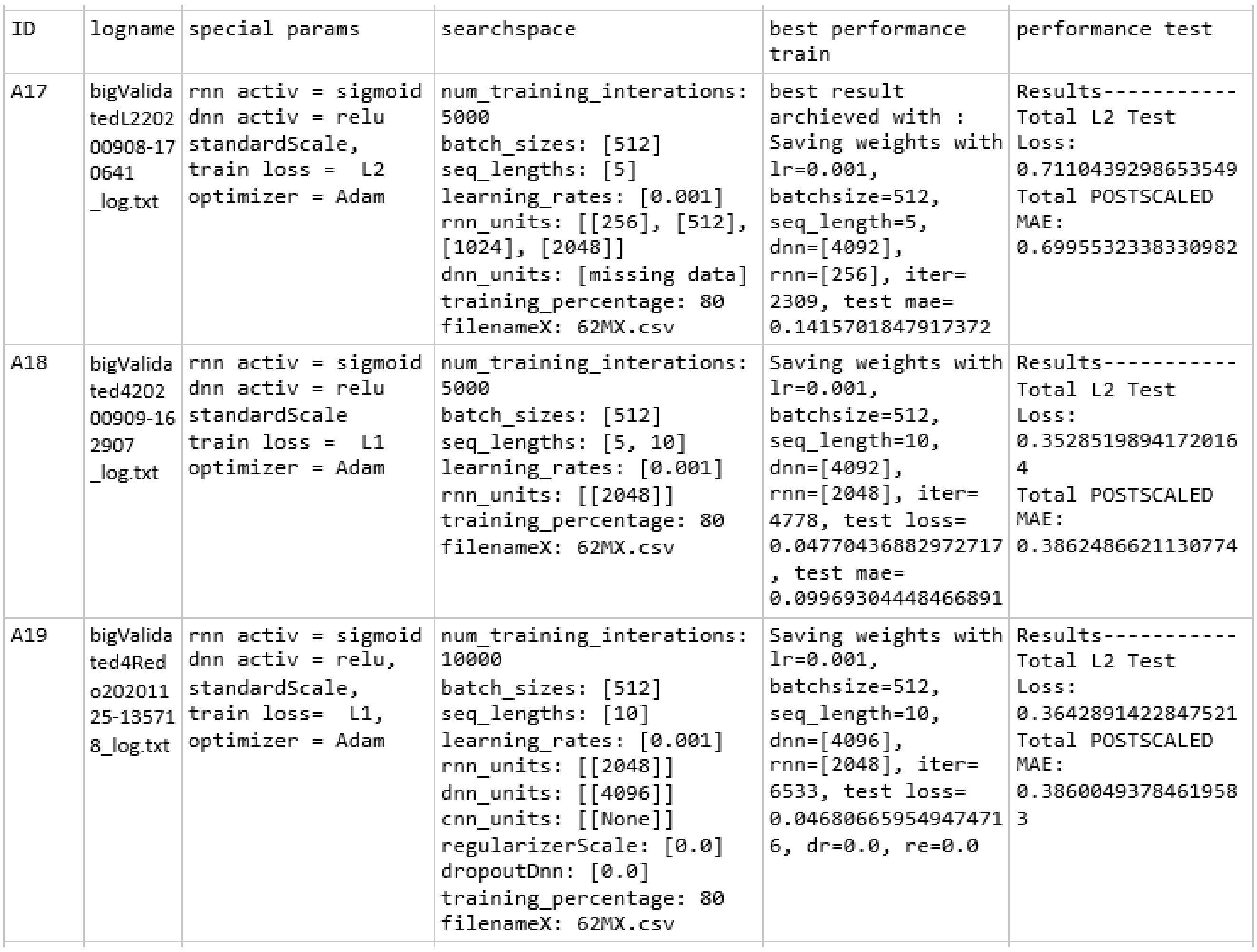
ID	logname	special params	searchspace	best performance train	performance test
A8	bigWithCollisions20200929-125751_log.txt	rnn activ = sigmoid dnn activ = relu, standardScale, train loss = L1, optimizer = Adam, Collisions	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.001] rnn_units: [[1024]] dnn_units: [[1024, 1024]] training_percentage: 80 filenameX: 62MXC.csv	best result archieved with : Saving weights with lr=0.001, batchsize=512, seq_length=5, dnn=[1024, 1024], rnn=[2048], iter= 3922, test loss= 0.23618924617767334, dr=0.0, re=0.0	Results----------- Total L2 Test Loss: 0.732322147025255 Total POSTSCALED MAE: 0.589955435148442
A9	bigWithCollisions20201012-145427_log.txt	rnn activ = sigmoid dnn activ = relu standardScale train loss = L1 optimizer = Adam, Collisions	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.001] rnn_units: [[2048]] dnn_units: [[4092]] training_percentage: 80 filenameX: 62MXC.csv	Saving weights with lr=0.001, batchsize=512, seq_length=5, dnn=[4092], rnn=[2048], iter= 2083, test loss= 0.22516626119613647	Results----------- Total L2 Test Loss: 0.621752498891304 Total POSTSCALED MAE: 0.548587959642613
A10	big-1,1e20200921-121728_log.txt	rnn activ = sigmoid dnn activ = relu MinMaxScale -1,1 train loss = L1 optimizer = Adam	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.001] rnn_units: [[2048]] dnn_units: [[4092]] training_percentage: 80 filenameX: 62MX.csv	Saving weights with lr=0.001, batchsize=512, seq_length=5, dnn=[4096], rnn=[2048], iter= 3844, test loss= 0.024765431880950928, dr=0.0, re=0.0	Results--------- Total L2 Test Loss: 0.248009710480040 Total POSTSCALED MAE: 6.341839664391273
A11	bigNoDnnMinMax20200913-161135_log.txt	rnn activ = sigmoid dnn activ = relu MinMaxScale 0,1 train loss = L1 optimizer = Adam	num_training_interations: 5000 batch_sizes: [2048] seq_lengths: [5] learning_rates: [0.001] rnn_units: [[128], [256], [512], [1024], [2048]] training_percentage: 80 filenameX: 62MX.csv	Saving weights with lr=0.001, batchsize=2048, seq_length=5, dnn=None, rnn=[128], iter= 4019, test mae= 0.28078426798492906	Results----------- Total L2 Test Loss: 0.12050233581007058 Total POSTSCALED MAE: 4.423971519993001
A12	bigDnnHeuristicSig20200915-121823_log.txt	rnn activ = sigmoid dnn activ = sigmoid standardScale NoRNN train loss = L1 optimizer = Adam	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [1] learning_rates: [0.001] rnn_units: [None] dnn_units=[[72],[96],[96*2+1],[72*5],[96*5],[5*96*2+1]]	Saving weights with lr=0.001, batchsize=512, seq_length=1, dnn=[193], rnn=None, iter= 2349, test loss= 0.05499543994665146, test mae= 0.10473784731762963	Results----------- Total L2 Test Loss: 0.512478110288828 Total POSTSCALED MAE: 0.402262931772767

Figure B.3.: LogsA08-12



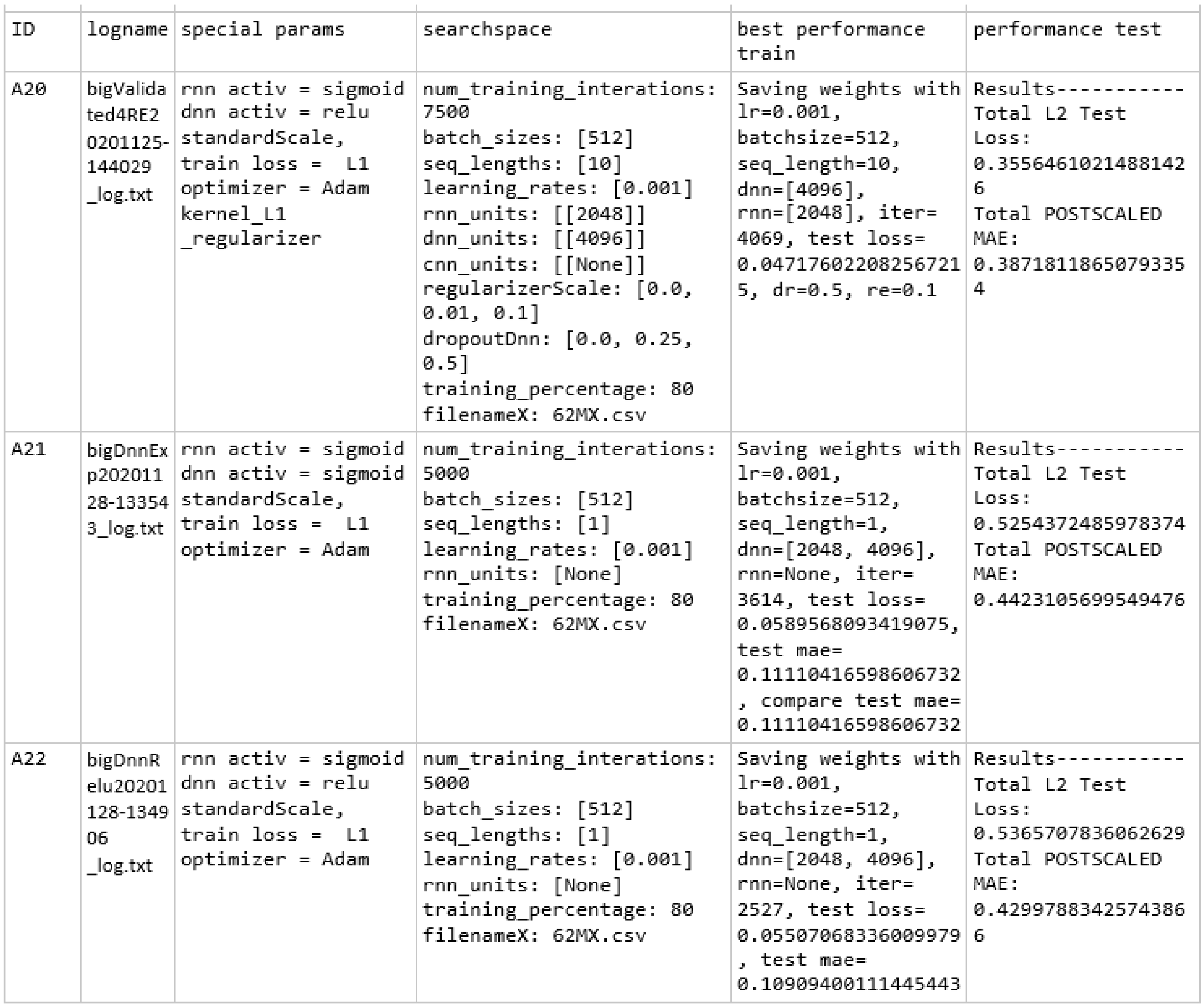
ID	logname	special params	searchspace	best performance train	performance test
A13	bigHeuristicBased20200915-100217_log.txt	rnn activ = sigmoid dnn activ = relu standardScale rnn layer based on heuristic train loss = L1 optimizer = Adam	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [5, 10] learning_rates: [0.001] rnn_units: [[96], [192], [480]] dnn_units: [[128], [256], [256, 256], [512], [512, 512], [1024], [1024, 1024], [2048], [4092]] training_percentage: 80 filenameX: 62MX.csv	Saving weights with lr=0.001, batchsize=512, seq_length=10, dnn=[256], rnn=[96], iter= 3085, test loss= 0.052007440477609634, test mae= 0.10275377738871679, compare test mae0.10275377738871679	Results----------- Total L2 Test Loss: 0.456398220304601 Total POSTSCALED MAE: 0.468345932934258
A14	bigV220200914-092210_log.txt	rnn activ = sigmoid dnn activ = relu standardScale train loss = L1 optimizer = Adam Velocities	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [2, 5, 8] learning_rates: [0.001] rnn_units: [[128], [256], [512], [1024], [2048]] dnn_units: [[128], [256], [256, 256], [512], [512, 512], [1024], [1024, 1024], [2048], [4092]] training_percentage: 80 filenameX: 62MVX2.csv	Saving weights with lr=0.001, batchsize=512, seq_length=5, dnn=[4092], rnn=[1024], iter= 3899, test loss= 0.05154065787792206, test mae= 0.09752753225908695	Results----------- Total L2 Test Loss: 1.9742666506030506 Total POSTSCALED MAE: 1.7154794994028228
A15	bigDnnlowD420201018-184521_log.txt	rnn activ = sigmoid dnn activ = relu standardScale, train loss = L1 optimizer = Adam	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [1] learning_rates: [0.001] rnn_units: [None] dnn_units=[[4],[16],[32],[64],[96],[96,96],[96,96,96],[96,96,96,96],[64,64],[64,64,64],[64,64,64,64],[32,32],[32,32,32],[32,32,32,32]] training_percentage: 80 filenameX: 62MX.csv	Saving weights with lr=0.001, batchsize=512, seq_length=1, dnn=[96], rnn=None, iter=4417, test loss= 0.2142263948917389, test mae= 0.3058735724210942,	Results----------- Total L2 Test Loss: 0.5278683263671756 Total POSTSCALED MAE: 0.4546908423073005
A16	bigLoaderGood20201018-183115_log.txt	rnn activ = sigmoid dnn activ = relu standardScale, train loss = L1 optimizer = Adam	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.001] rnn_units: [[2048]] dnn_units: [[4092]] training_percentage: 80 filenameX: 62MX.csv	Saving weights with lr=0.001, batchsize=512, seq_length=5, dnn=[4092], rnn=[2048], iter= 3502, test loss= 0.051248230040073395	Results----------- Total L2 Test Loss: 0.43982158499104274 Total POSTSCALED MAE: 0.4306260851573251

Figure B.4.: LogsA13-16



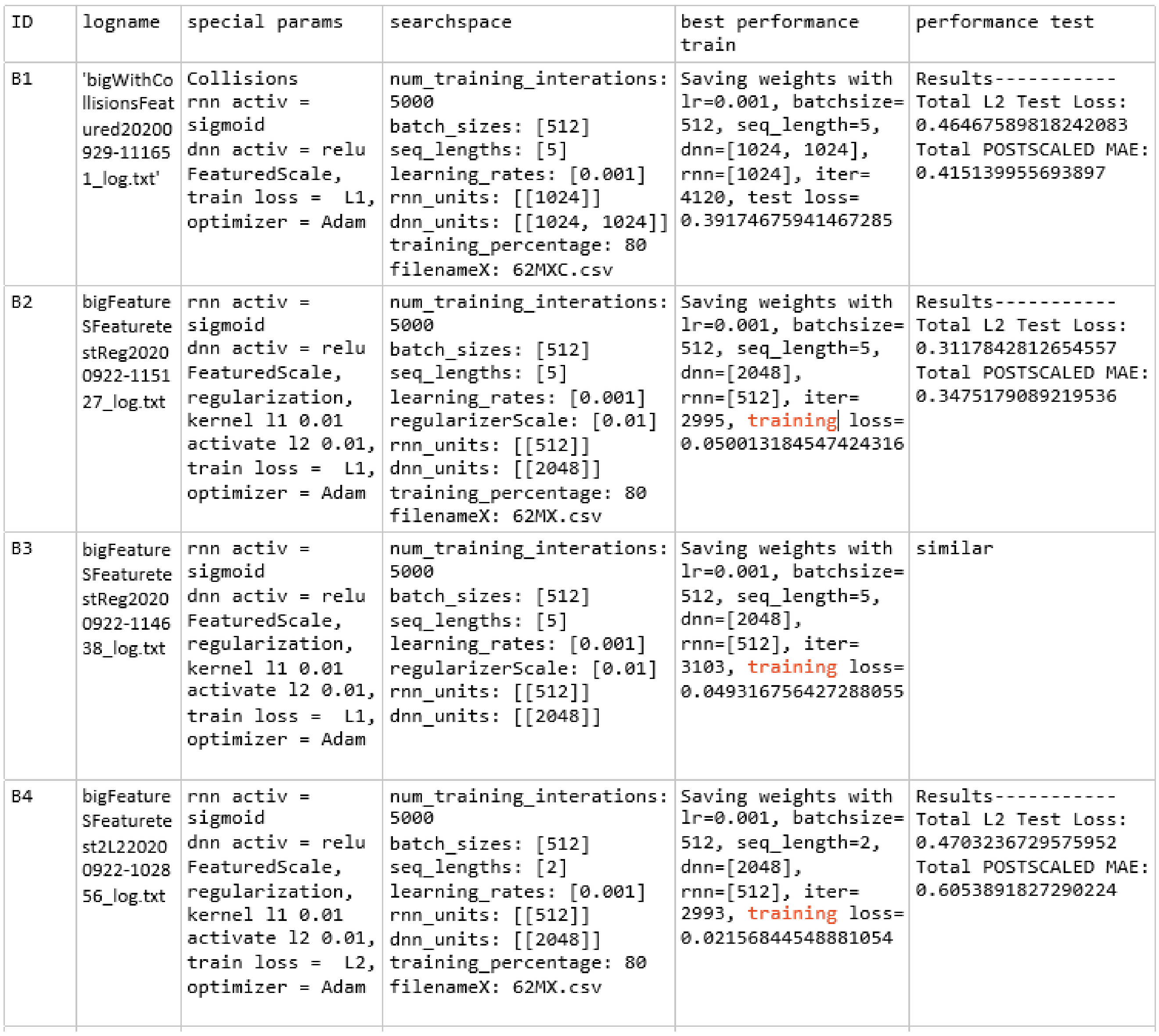
ID	logname	special params	searchspace	best performance train	performance test
A17	bigValidatedL220200908-170641_log.txt	rnn activ = sigmoid dnn activ = relu standardScale, train loss = L2 optimizer = Adam	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.001] rnn_units: [[256], [512], [1024], [2048]] dnn_units: [missing data] training_percentage: 80 filenameX: 62MX.csv	best result archieved with : Saving weights with lr=0.001, batchsize=512, seq_length=5, dnn=[4092], rnn=[256], iter= 2309, test mae= 0.1415701847917372	Results----------- Total L2 Test Loss: 0.7110439298653549 Total POSTSCALED MAE: 0.6995532338330982
A18	bigValidated420200909-162907_log.txt	rnn activ = sigmoid dnn activ = relu standardScale train loss = L1 optimizer = Adam	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [5, 10] learning_rates: [0.001] rnn_units: [[2048]] training_percentage: 80 filenameX: 62MX.csv	Saving weights with lr=0.001, batchsize=512, seq_length=10, dnn=[4092], rnn=[2048], iter= 4778, test loss= 0.04770436882972717, test mae= 0.09969304448466891	Results----------- Total L2 Test Loss: 0.35285198941720164 Total POSTSCALED MAE: 0.3862486621130774
A19	bigValidated4Redo20201125-135718_log.txt	rnn activ = sigmoid dnn activ = relu, standardScale, train loss= L1, optimizer = Adam	num_training_interations: 10000 batch_sizes: [512] seq_lengths: [10] learning_rates: [0.001] rnn_units: [[2048]] dnn_units: [[4096]] cnn_units: [[None]] regularizerScale: [0.0] dropoutDnn: [0.0] training_percentage: 80 filenameX: 62MX.csv	Saving weights with lr=0.001, batchsize=512, seq_length=10, dnn=[4096], rnn=[2048], iter= 6533, test loss= 0.046806659549474716, dr=0.0, re=0.0	Results----------- Total L2 Test Loss: 0.3642891422847521 Total POSTSCALED MAE: 0.38600493784619583

Figure B.5.: LogsA17-19



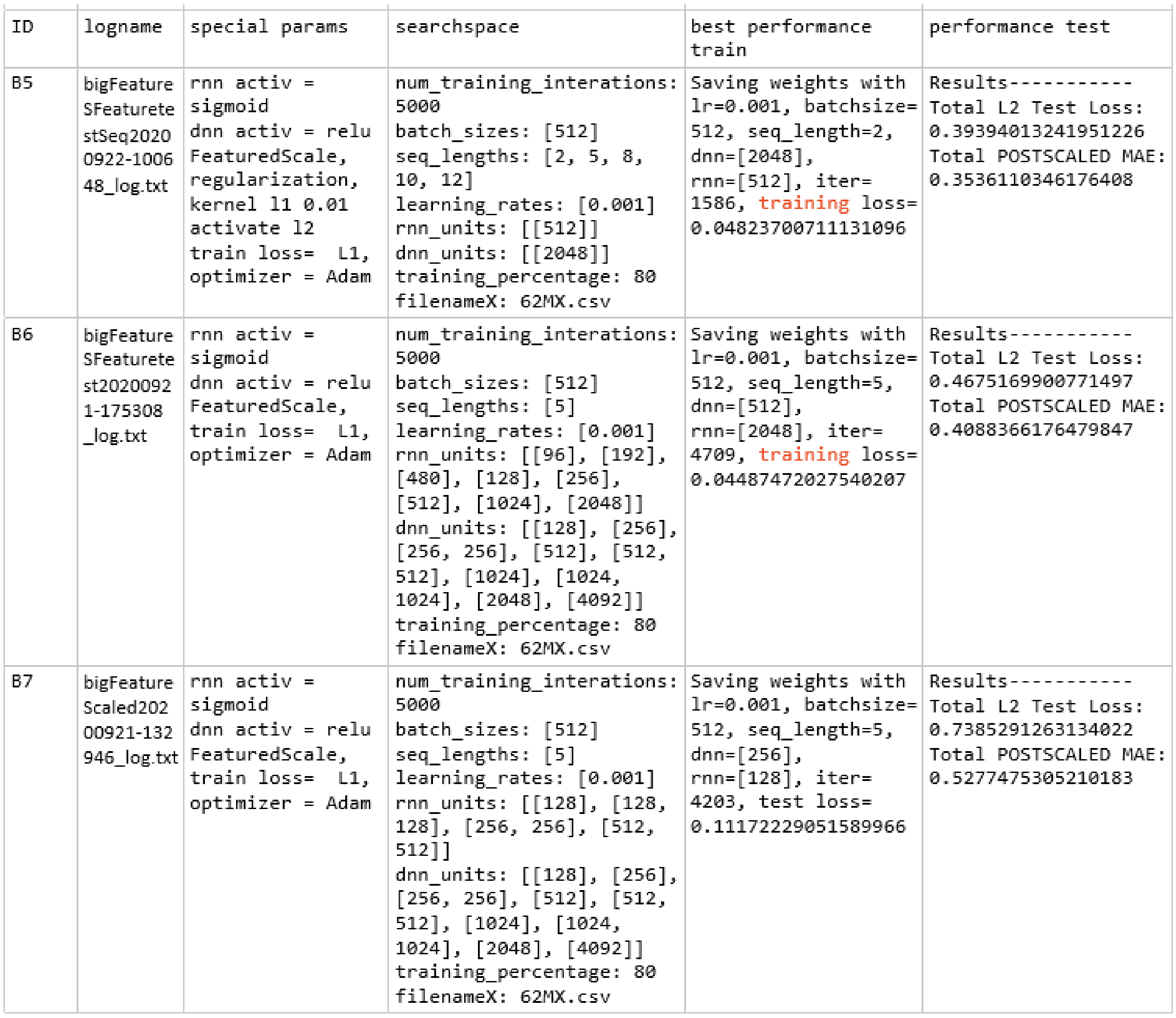
ID	logname	special params	searchspace	best performance train	performance test
A20	bigValidated4RE20201125-144029_log.txt	rnn activ = sigmoid dnn activ = relu standardScale, train loss = L1 optimizer = Adam kernel_L1_regularizer	num_training_interations: 7500 batch_sizes: [512] seq_lengths: [10] learning_rates: [0.001] rnn_units: [[2048]] dnn_units: [[4096]] cnn_units: [[None]] regularizerScale: [0.0, 0.01, 0.1] dropoutDnn: [0.0, 0.25, 0.5] training_percentage: 80 filenameX: 62MX.csv	Saving weights with lr=0.001, batchsize=512, seq_length=10, dnn=[4096], rnn=[2048], iter=4069, test loss=0.047176022082567215, dr=0.5, re=0.1	Results----------- Total L2 Test Loss: 0.35564610214881426 Total POSTSCALED MAE: 0.38718118650793354
A21	bigDnnExp20201128-133543_log.txt	rnn activ = sigmoid dnn activ = sigmoid standardScale, train loss = L1 optimizer = Adam	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [1] learning_rates: [0.001] rnn_units: [None] training_percentage: 80 filenameX: 62MX.csv	Saving weights with lr=0.001, batchsize=512, seq_length=1, dnn=[2048, 4096], rnn=None, iter=3614, test loss=0.0589568093419075, test mae=0.11110416598606732, compare test mae=0.11110416598606732	Results----------- Total L2 Test Loss: 0.5254372485978374 Total POSTSCALED MAE: 0.4423105699549476
A22	bigDnnRelu20201128-134906_log.txt	rnn activ = sigmoid dnn activ = relu standardScale, train loss = L1 optimizer = Adam	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [1] learning_rates: [0.001] rnn_units: [None] training_percentage: 80 filenameX: 62MX.csv	Saving weights with lr=0.001, batchsize=512, seq_length=1, dnn=[2048, 4096], rnn=None, iter=2527, test loss=0.05507068336009979, test mae=0.10909400111445443	Results----------- Total L2 Test Loss: 0.5365707836062629 Total POSTSCALED MAE: 0.42997883425743866

Figure B.6.: LogsA20-22



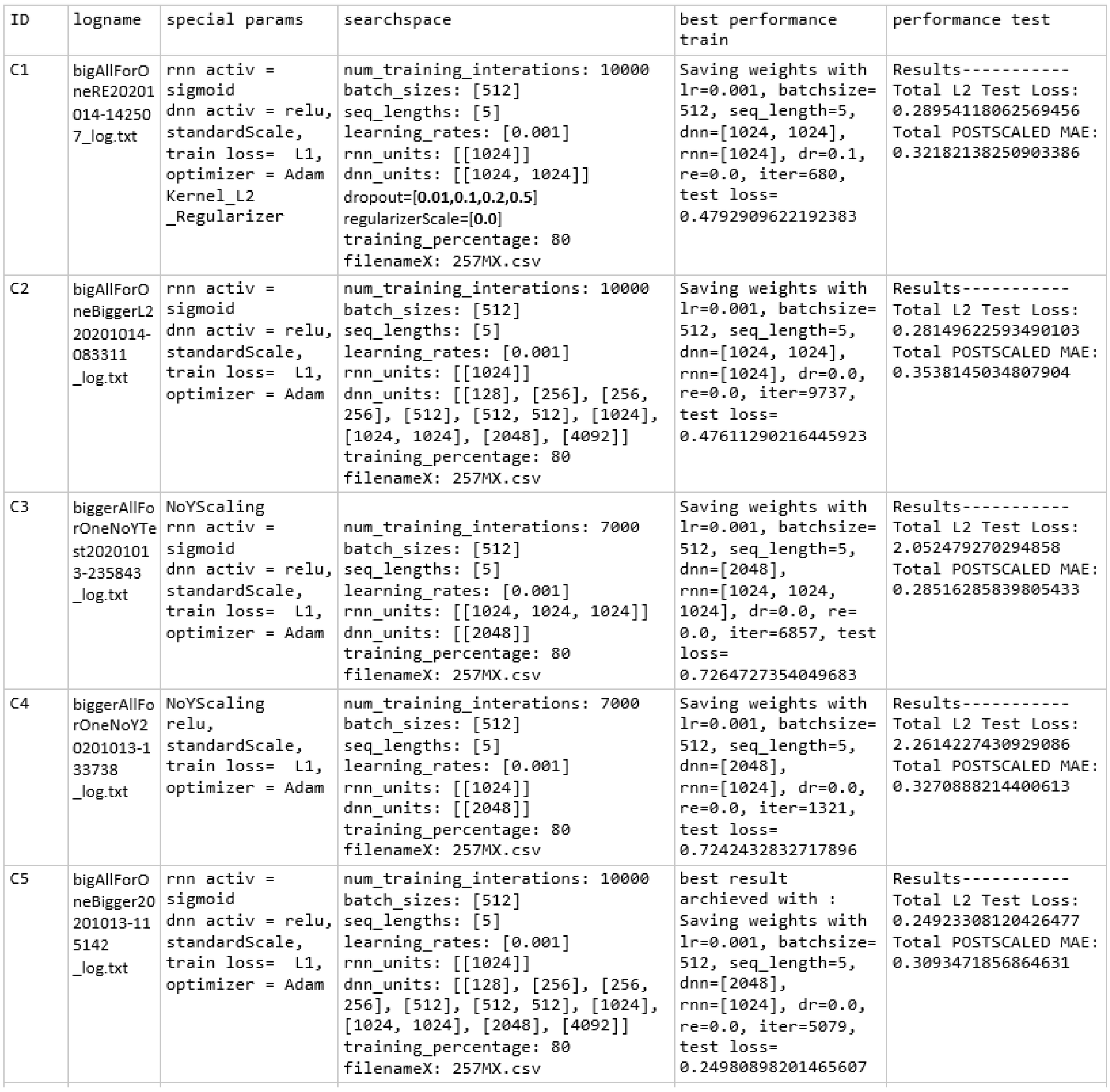
ID	logname	special params	searchspace	best performance train	performance test
B1	'bigWithCollisionsFeatured20200929-111651_log.txt'	Collisions rnn activ = sigmoid dnn activ = relu FeaturedScale, train loss = L1, optimizer = Adam	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.001] rnn_units: [[1024]] dnn_units: [[1024, 1024]] training_percentage: 80 filenameX: 62MXC.csv	Saving weights with lr=0.001, batchsize=512, seq_length=5, dnn=[1024, 1024], rnn=[1024], iter=4120, test loss=0.39174675941467285	Results----------- Total L2 Test Loss: 0.46467589818242083 Total POSTSCALED MAE: 0.415139955693897
B2	bigFeatureSFeaturestReg20200922-115127_log.txt	rnn activ = sigmoid dnn activ = relu FeaturedScale, regularization, kernel l1 0.01 activate l2 0.01, train loss = L1, optimizer = Adam	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.001] regularizerScale: [0.01] rnn_units: [[512]] dnn_units: [[2048]] training_percentage: 80 filenameX: 62MX.csv	Saving weights with lr=0.001, batchsize=512, seq_length=5, dnn=[2048], rnn=[512], iter=2995, training loss=0.050013184547424316	Results----------- Total L2 Test Loss: 0.3117842812654557 Total POSTSCALED MAE: 0.3475179089219536
B3	bigFeatureSFeaturestReg20200922-114638_log.txt	rnn activ = sigmoid dnn activ = relu FeaturedScale, regularization, kernel l1 0.01 activate l2 0.01, train loss = L1, optimizer = Adam	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.001] regularizerScale: [0.01] rnn_units: [[512]] dnn_units: [[2048]]	Saving weights with lr=0.001, batchsize=512, seq_length=5, dnn=[2048], rnn=[512], iter=3103, training loss=0.049316756427288055	similar
B4	bigFeatureSFeaturest2L220200922-102856_log.txt	rnn activ = sigmoid dnn activ = relu FeaturedScale, regularization, kernel l1 0.01 activate l2 0.01, train loss = L2, optimizer = Adam	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [2] learning_rates: [0.001] rnn_units: [[512]] dnn_units: [[2048]] training_percentage: 80 filenameX: 62MX.csv	Saving weights with lr=0.001, batchsize=512, seq_length=2, dnn=[2048], rnn=[512], iter=2993, training loss=0.02156844548881054	Results----------- Total L2 Test Loss: 0.4703236729575952 Total POSTSCALED MAE: 0.6053891827290224

Figure B.7.: LogsB01-04



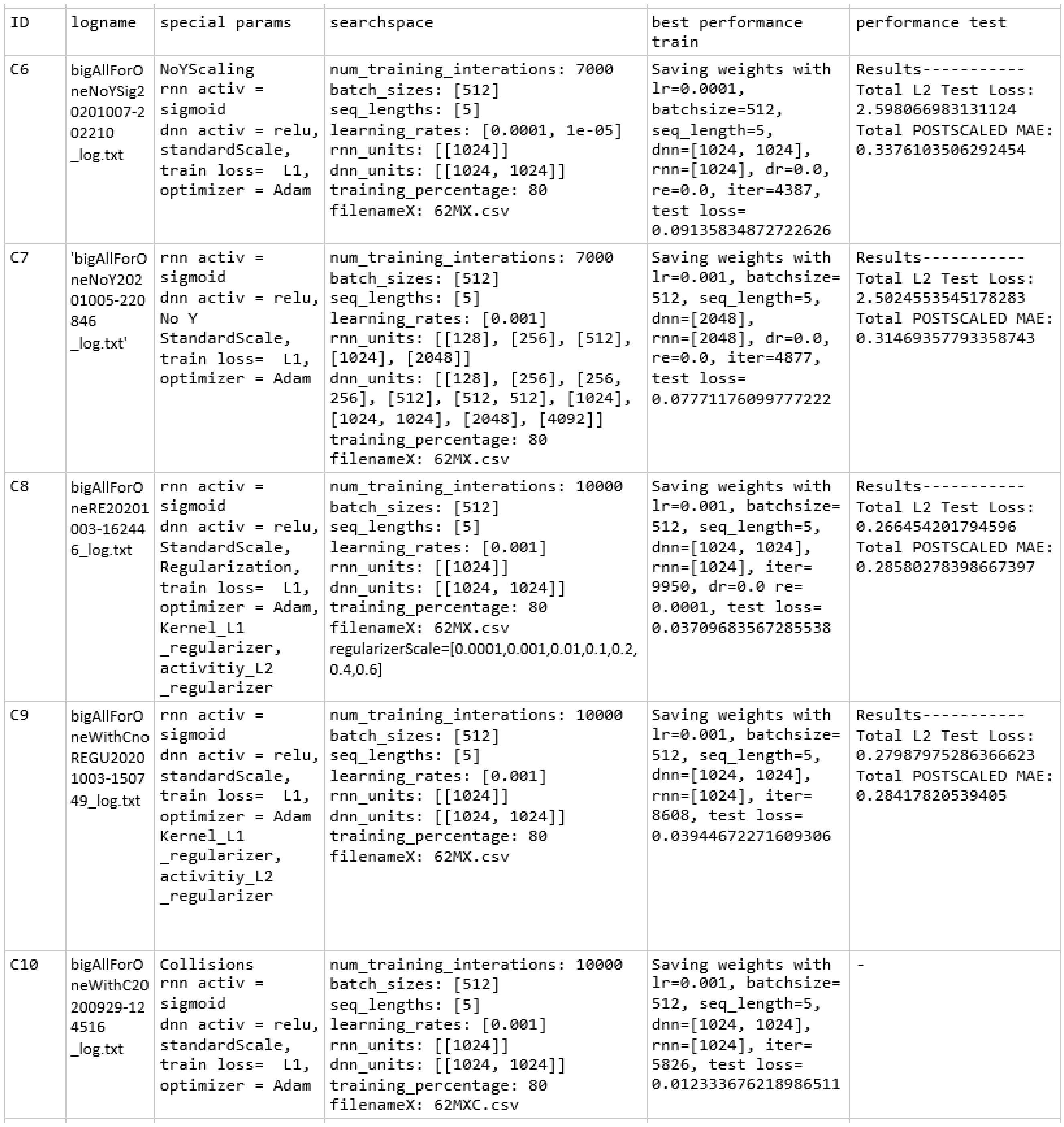
ID	logname	special params	searchspace	best performance train	performance test
B5	bigFeatureSFeaturetestSeq20200922-100648_log.txt	rnn activ = sigmoid dnn activ = relu FeaturedScale, regularization, kernel l1 0.01 activate l2 train loss= L1, optimizer = Adam	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [2, 5, 8, 10, 12] learning_rates: [0.001] rnn_units: [[512]] dnn_units: [[2048]] training_percentage: 80 filenameX: 62MX.csv	Saving weights with lr=0.001, batchsize=512, seq_length=2, dnn=[2048], rnn=[512], iter=1586, training loss=0.04823700711131096	Results----------- Total L2 Test Loss: 0.39394013241951226 Total POSTSCALED MAE: 0.3536110346176408
B6	bigFeatureSFeaturetest20200921-175308_log.txt	rnn activ = sigmoid dnn activ = relu FeaturedScale, train loss= L1, optimizer = Adam	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.001] rnn_units: [[96], [192], [480], [128], [256], [512], [1024], [2048]] dnn_units: [[128], [256], [256, 256], [512], [512, 512], [1024], [1024, 1024], [2048], [4092]] training_percentage: 80 filenameX: 62MX.csv	Saving weights with lr=0.001, batchsize=512, seq_length=5, dnn=[512], rnn=[2048], iter=4709, training loss=0.04487472027540207	Results----------- Total L2 Test Loss: 0.4675169900771497 Total POSTSCALED MAE: 0.4088366176479847
B7	bigFeatureScaled20200921-132946_log.txt	rnn activ = sigmoid dnn activ = relu FeaturedScale, train loss= L1, optimizer = Adam	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.001] rnn_units: [[128], [128, 128], [256, 256], [512, 512]] dnn_units: [[128], [256], [256, 256], [512], [512, 512], [1024], [1024, 1024], [2048], [4092]] training_percentage: 80 filenameX: 62MX.csv	Saving weights with lr=0.001, batchsize=512, seq_length=5, dnn=[256], rnn=[128], iter=4203, test loss=0.11172229051589966	Results----------- Total L2 Test Loss: 0.7385291263134022 Total POSTSCALED MAE: 0.5277475305210183

Figure B.8.: LogsB05-07



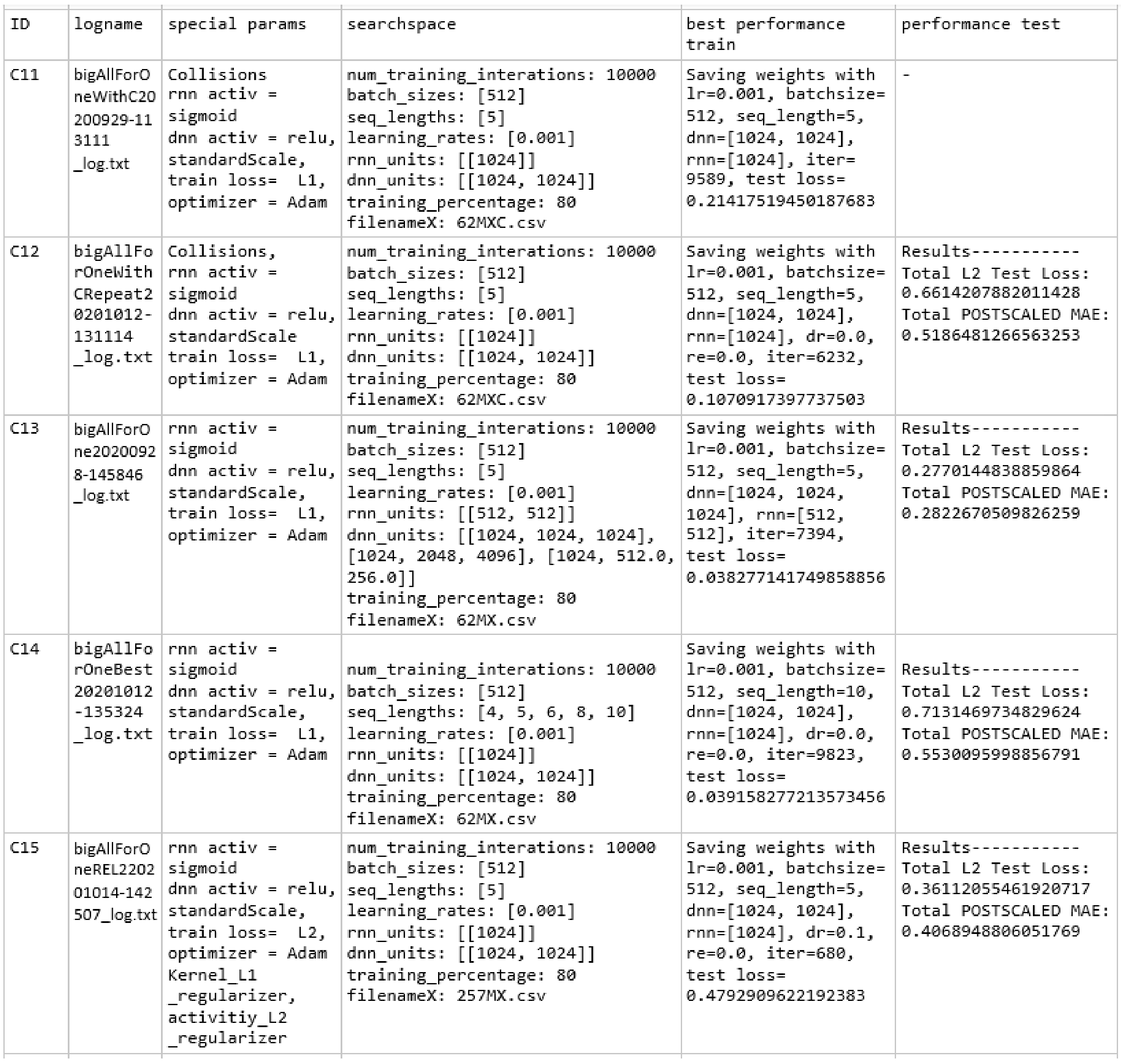
ID	logname	special params	searchspace	best performance train	performance test
C1	bigAllForOneRE20201014-142507_log.txt	rnn activ = sigmoid dnn activ = relu, standardScale, train loss= L1, optimizer = Adam Kernel_L2 _Regularizer	num_training_interations: 10000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.001] rnn_units: [[1024]] dnn_units: [[1024, 1024]] dropout=[**0.01,0.1,0.2,0.5**] regularizerScale=[**0.0**] training_percentage: 80 filenameX: 257MX.csv	Saving weights with lr=0.001, batchsize=512, seq_length=5, dnn=[1024, 1024], rnn=[1024], dr=0.1, re=0.0, iter=680, test loss= 0.4792909622192383	Results----------- Total L2 Test Loss: 0.28954118062569456 Total POSTSCALED MAE: 0.32182138250903386
C2	bigAllForOneBiggerL220201014-083311 _log.txt	rnn activ = sigmoid dnn activ = relu, standardScale, train loss= L1, optimizer = Adam	num_training_interations: 10000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.001] rnn_units: [[1024]] dnn_units: [[128], [256], [256, 256], [512], [512, 512], [1024], [1024, 1024], [2048], [4092]] training_percentage: 80 filenameX: 257MX.csv	Saving weights with lr=0.001, batchsize=512, seq_length=5, dnn=[1024, 1024], rnn=[1024], dr=0.0, re=0.0, iter=9737, test loss= 0.47611290216445923	Results----------- Total L2 Test Loss: 0.28149622593490103 Total POSTSCALED MAE: 0.3538145034807904
C3	biggerAllForOneNoYTest20201013-235843 _log.txt	NoYScaling rnn activ = sigmoid dnn activ = relu, standardScale, train loss= L1, optimizer = Adam	num_training_interations: 7000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.001] rnn_units: [[1024, 1024, 1024]] dnn_units: [[2048]] training_percentage: 80 filenameX: 257MX.csv	Saving weights with lr=0.001, batchsize=512, seq_length=5, dnn=[2048], rnn=[1024, 1024, 1024], dr=0.0, re=0.0, iter=6857, test loss= 0.7264727354049683	Results----------- Total L2 Test Loss: 2.052479270294858 Total POSTSCALED MAE: 0.28516285839805433
C4	biggerAllForOneNoY20201013-133738 _log.txt	NoYScaling relu, standardScale, train loss= L1, optimizer = Adam	num_training_interations: 7000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.001] rnn_units: [[1024]] dnn_units: [[2048]] training_percentage: 80 filenameX: 257MX.csv	Saving weights with lr=0.001, batchsize=512, seq_length=5, dnn=[2048], rnn=[1024], dr=0.0, re=0.0, iter=1321, test loss= 0.7242432832717896	Results----------- Total L2 Test Loss: 2.2614227430929086 Total POSTSCALED MAE: 0.3270888214400613
C5	bigAllForOneBigger20201013-115142 _log.txt	rnn activ = sigmoid dnn activ = relu, standardScale, train loss= L1, optimizer = Adam	num_training_interations: 10000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.001] rnn_units: [[1024]] dnn_units: [[128], [256], [256, 256], [512], [512, 512], [1024], [1024, 1024], [2048], [4092]] training_percentage: 80 filenameX: 257MX.csv	best result archieved with : Saving weights with lr=0.001, batchsize=512, seq_length=5, dnn=[2048], rnn=[1024], dr=0.0, re=0.0, iter=5079, test loss= 0.24980898201465607	Results----------- Total L2 Test Loss: 0.24923308120426477 Total POSTSCALED MAE: 0.3093471856864631

Figure B.9.: LogsC01-05



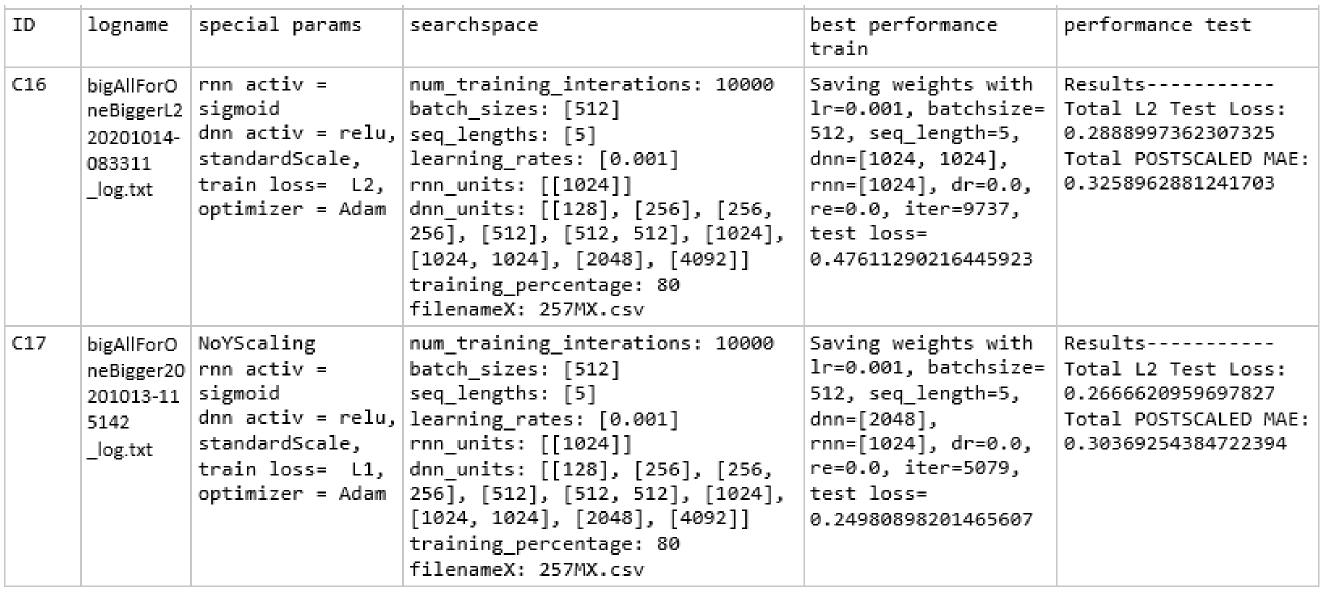
ID	logname	special params	searchspace	best performance train	performance test
C6	bigAllForOneNoYSig20201007-202210_log.txt	NoYScaling rnn activ = sigmoid dnn activ = relu, standardScale, train loss= L1, optimizer = Adam	num_training_interations: 7000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.0001, 1e-05] rnn_units: [[1024]] dnn_units: [[1024, 1024]] training_percentage: 80 filenameX: 62MX.csv	Saving weights with lr=0.0001, batchsize=512, seq_length=5, dnn=[1024, 1024], rnn=[1024], dr=0.0, re=0.0, iter=4387, test loss= 0.09135834872722626	Results----------- Total L2 Test Loss: 2.598066983131124 Total POSTSCALED MAE: 0.3376103506292454
C7	'bigAllForOneNoY20201005-220846_log.txt'	rnn activ = sigmoid dnn activ = relu, No Y StandardScale, train loss= L1, optimizer = Adam	num_training_interations: 7000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.001] rnn_units: [[128], [256], [512], [1024], [2048]] dnn_units: [[128], [256], [256, 256], [512], [512, 512], [1024], [1024, 1024], [2048], [4092]] training_percentage: 80 filenameX: 62MX.csv	Saving weights with lr=0.001, batchsize= 512, seq_length=5, dnn=[2048], rnn=[2048], dr=0.0, re=0.0, iter=4877, test loss= 0.07771176099777222	Results----------- Total L2 Test Loss: 2.5024553545178283 Total POSTSCALED MAE: 0.31469357793358743
C8	bigAllForOneRE20201003-162446_log.txt	rnn activ = sigmoid dnn activ = relu, StandardScale, Regularization, train loss= L1, optimizer = Adam, Kernel_L1 _regularizer, activitiy_L2 _regularizer	num_training_interations: 10000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.001] rnn_units: [[1024]] dnn_units: [[1024, 1024]] training_percentage: 80 filenameX: 62MX.csv regularizerScale=[0.0001,0.001,0.01,0.1,0.2, 0.4,0.6]	Saving weights with lr=0.001, batchsize= 512, seq_length=5, dnn=[1024, 1024], rnn=[1024], iter= 9950, dr=0.0 re= 0.0001, test loss= 0.03709683567285538	Results----------- Total L2 Test Loss: 0.266454201794596 Total POSTSCALED MAE: 0.28580278398667397
C9	bigAllForOneWithCnoREGU20201003-150749_log.txt	rnn activ = sigmoid dnn activ = relu, standardScale, train loss= L1, optimizer = Adam Kernel_L1 _regularizer, activitiy_L2 _regularizer	num_training_interations: 10000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.001] rnn_units: [[1024]] dnn_units: [[1024, 1024]] training_percentage: 80 filenameX: 62MX.csv	Saving weights with lr=0.001, batchsize= 512, seq_length=5, dnn=[1024, 1024], rnn=[1024], iter= 8608, test loss= 0.03944672271609306	Results----------- Total L2 Test Loss: 0.27987975286366623 Total POSTSCALED MAE: 0.28417820539405
C10	bigAllForOneWithC20200929-124516 _log.txt	Collisions rnn activ = sigmoid dnn activ = relu, standardScale, train loss= L1, optimizer = Adam	num_training_interations: 10000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.001] rnn_units: [[1024]] dnn_units: [[1024, 1024]] training_percentage: 80 filenameX: 62MXC.csv	Saving weights with lr=0.001, batchsize= 512, seq_length=5, dnn=[1024, 1024], rnn=[1024], iter= 5826, test loss= 0.012333676218986511	-

Figure B.10.: LogsC06-10



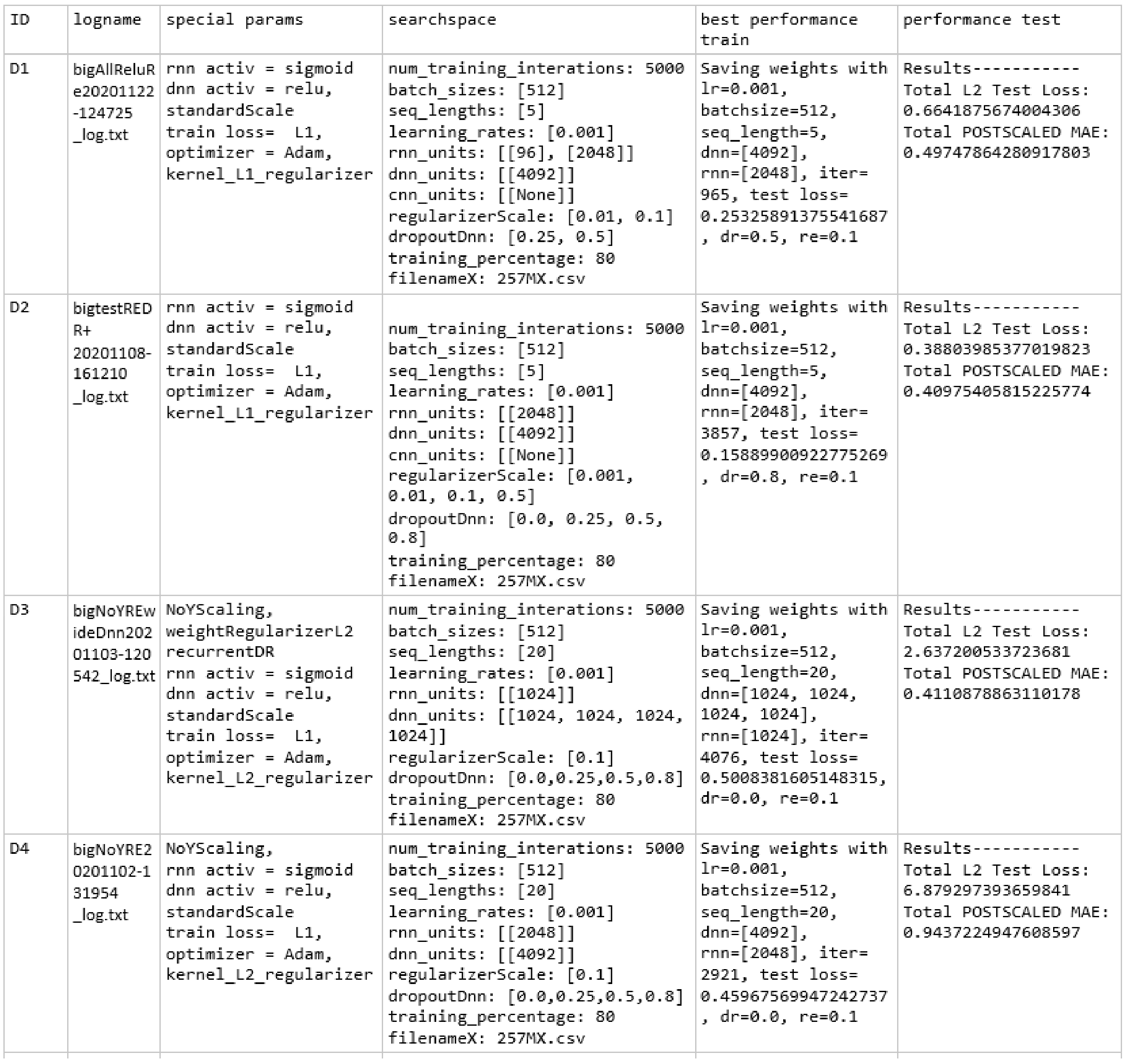
ID	logname	special params	searchspace	best performance train	performance test
C11	bigAllForOneWithC20200929-113111_log.txt	Collisions rnn activ = sigmoid dnn activ = relu, standardScale, train loss= L1, optimizer = Adam	num_training_interations: 10000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.001] rnn_units: [[1024]] dnn_units: [[1024, 1024]] training_percentage: 80 filenameX: 62MXC.csv	Saving weights with lr=0.001, batchsize=512, seq_length=5, dnn=[1024, 1024], rnn=[1024], iter=9589, test loss=0.21417519450187683	-
C12	bigAllForOneWithCRepeat20201012-131114_log.txt	Collisions, rnn activ = sigmoid dnn activ = relu, standardScale train loss= L1, optimizer = Adam	num_training_interations: 10000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.001] rnn_units: [[1024]] dnn_units: [[1024, 1024]] training_percentage: 80 filenameX: 62MXC.csv	Saving weights with lr=0.001, batchsize=512, seq_length=5, dnn=[1024, 1024], rnn=[1024], dr=0.0, re=0.0, iter=6232, test loss=0.1070917397737503	Results----------- Total L2 Test Loss: 0.6614207882011428 Total POSTSCALED MAE: 0.5186481266563253
C13	bigAllForOne20200928-145846_log.txt	rnn activ = sigmoid dnn activ = relu, standardScale, train loss= L1, optimizer = Adam	num_training_interations: 10000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.001] rnn_units: [[512, 512]] dnn_units: [[1024, 1024, 1024], [1024, 2048, 4096], [1024, 512.0, 256.0]] training_percentage: 80 filenameX: 62MX.csv	Saving weights with lr=0.001, batchsize=512, seq_length=5, dnn=[1024, 1024, 1024], rnn=[512, 512], iter=7394, test loss=0.038277141749858856	Results----------- Total L2 Test Loss: 0.2770144838859864 Total POSTSCALED MAE: 0.2822670509826259
C14	bigAllForOneBest20201012-135324_log.txt	rnn activ = sigmoid dnn activ = relu, standardScale, train loss= L1, optimizer = Adam	num_training_interations: 10000 batch_sizes: [512] seq_lengths: [4, 5, 6, 8, 10] learning_rates: [0.001] rnn_units: [[1024]] dnn_units: [[1024, 1024]] training_percentage: 80 filenameX: 62MX.csv	Saving weights with lr=0.001, batchsize=512, seq_length=10, dnn=[1024, 1024], rnn=[1024], dr=0.0, re=0.0, iter=9823, test loss=0.039158277213573456	Results----------- Total L2 Test Loss: 0.7131469734829624 Total POSTSCALED MAE: 0.5530095998856791
C15	bigAllForOneREL20201014-142507_log.txt	rnn activ = sigmoid dnn activ = relu, standardScale, train loss= L2, optimizer = Adam Kernel_L1 _regularizer, activitiy_L2 _regularizer	num_training_interations: 10000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.001] rnn_units: [[1024]] dnn_units: [[1024, 1024]] training_percentage: 80 filenameX: 257MX.csv	Saving weights with lr=0.001, batchsize=512, seq_length=5, dnn=[1024, 1024], rnn=[1024], dr=0.1, re=0.0, iter=680, test loss=0.4792909622192383	Results----------- Total L2 Test Loss: 0.36112055461920717 Total POSTSCALED MAE: 0.4068948806051769

Figure B.11.: LogsC11-15



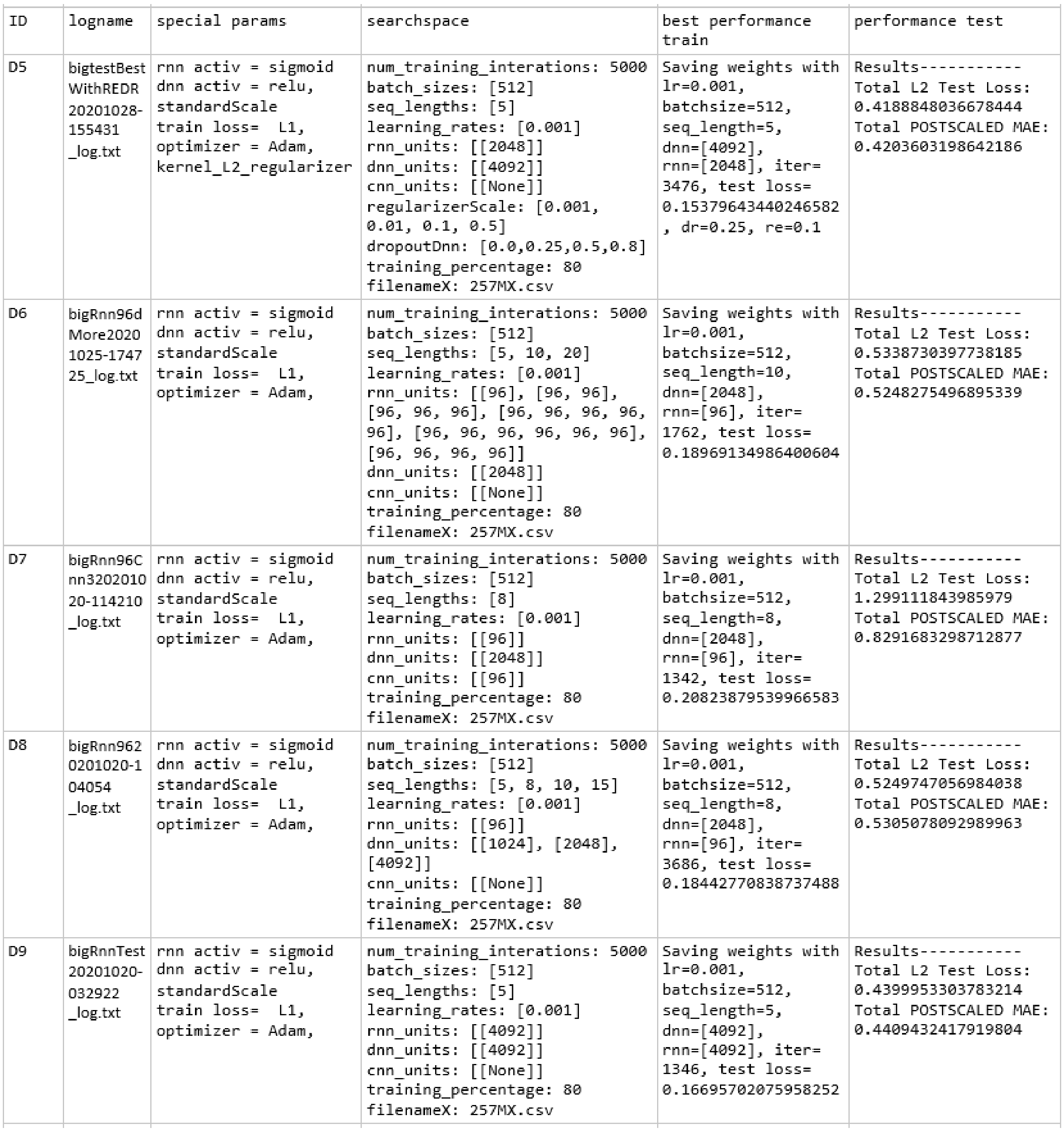
ID	logname	special params	searchspace	best performance train	performance test
C16	bigAllForO neBiggerL2 20201014- 083311 _log.txt	rnn activ = sigmoid dnn activ = relu, standardScale, train loss= L2, optimizer = Adam	num_training_interations: 10000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.001] rnn_units: [[1024]] dnn_units: [[128], [256], [256, 256], [512], [512, 512], [1024], [1024, 1024], [2048], [4092]] training_percentage: 80 filenameX: 257MX.csv	Saving weights with lr=0.001, batchsize= 512, seq_length=5, dnn=[1024, 1024], rnn=[1024], dr=0.0, re=0.0, iter=9737, test loss= 0.47611290216445923	Results----------- Total L2 Test Loss: 0.2888997362307325 Total POSTSCALED MAE: 0.3258962881241703
C17	bigAllForO neBigger20 201013-11 5142 _log.txt	NoYScaling rnn activ = sigmoid dnn activ = relu, standardScale, train loss= L1, optimizer = Adam	num_training_interations: 10000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.001] rnn_units: [[1024]] dnn_units: [[128], [256], [256, 256], [512], [512, 512], [1024], [1024, 1024], [2048], [4092]] training_percentage: 80 filenameX: 257MX.csv	Saving weights with lr=0.001, batchsize= 512, seq_length=5, dnn=[2048], rnn=[1024], dr=0.0, re=0.0, iter=5079, test loss= 0.24980898201465607	Results----------- Total L2 Test Loss: 0.2666620959697827 Total POSTSCALED MAE: 0.30369254384722394

Figure B.12.: LogsC16-17



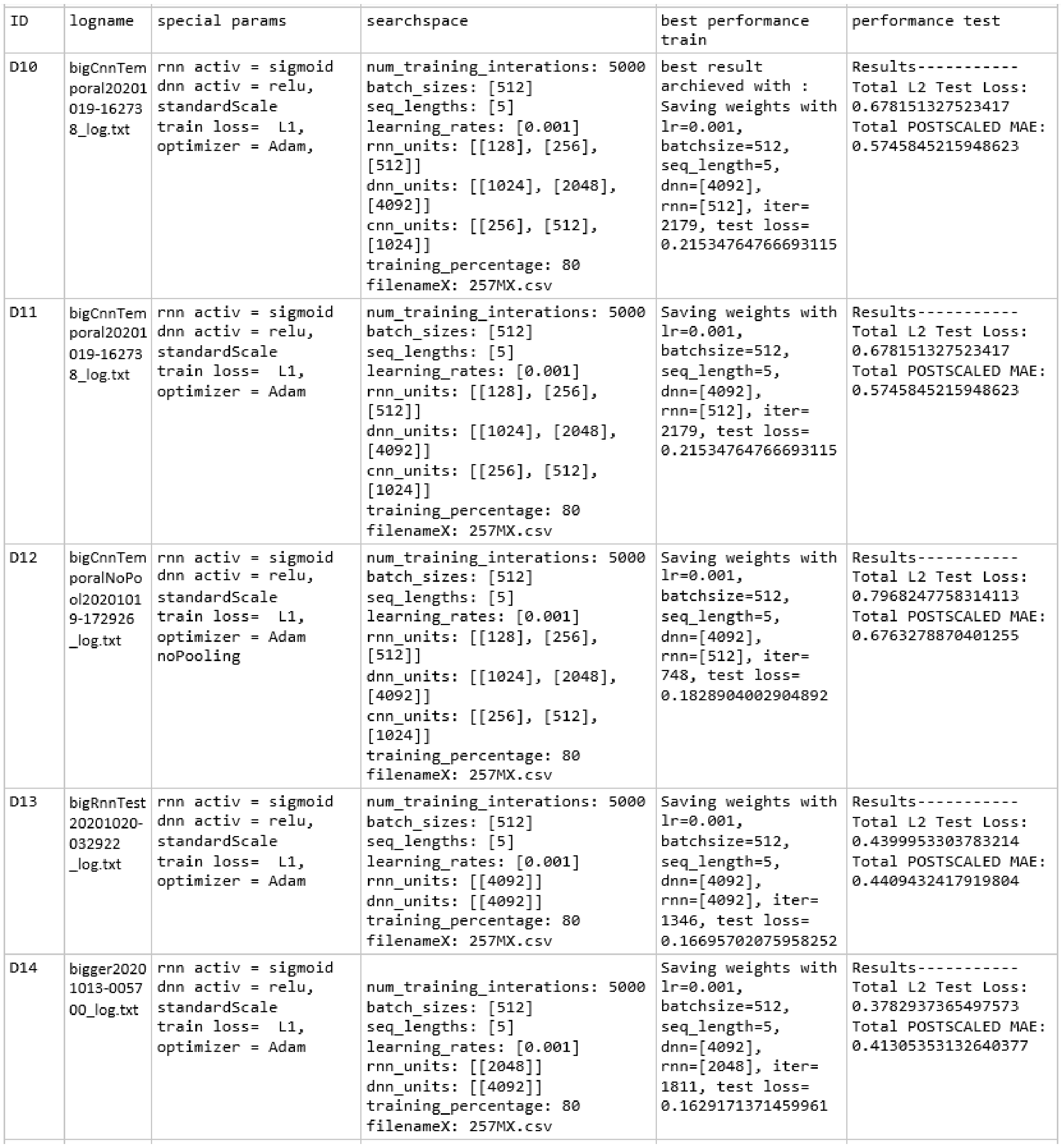
ID	logname	special params	searchspace	best performance train	performance test
D1	bigAllReluRe20201122-124725_log.txt	rnn activ = sigmoid dnn activ = relu, standardScale train loss= L1, optimizer = Adam, kernel_L1_regularizer	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.001] rnn_units: [[96], [2048]] dnn_units: [[4092]] cnn_units: [[None]] regularizerScale: [0.01, 0.1] dropoutDnn: [0.25, 0.5] training_percentage: 80 filenameX: 257MX.csv	Saving weights with lr=0.001, batchsize=512, seq_length=5, dnn=[4092], rnn=[2048], iter=965, test loss=0.25325891375541687, dr=0.5, re=0.1	Results----------- Total L2 Test Loss: 0.6641875674004306 Total POSTSCALED MAE: 0.49747864280917803
D2	bigtestREDR+20201108-161210_log.txt	rnn activ = sigmoid dnn activ = relu, standardScale train loss= L1, optimizer = Adam, kernel_L1_regularizer	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.001] rnn_units: [[2048]] dnn_units: [[4092]] cnn_units: [[None]] regularizerScale: [0.001, 0.01, 0.1, 0.5] dropoutDnn: [0.0, 0.25, 0.5, 0.8] training_percentage: 80 filenameX: 257MX.csv	Saving weights with lr=0.001, batchsize=512, seq_length=5, dnn=[4092], rnn=[2048], iter=3857, test loss=0.15889900922775269, dr=0.8, re=0.1	Results----------- Total L2 Test Loss: 0.38803985377019823 Total POSTSCALED MAE: 0.40975405815225774
D3	bigNoYREwideDnn20201103-120542_log.txt	NoYScaling, weightRegularizerL2 recurrentDR rnn activ = sigmoid dnn activ = relu, standardScale train loss= L1, optimizer = Adam, kernel_L2_regularizer	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [20] learning_rates: [0.001] rnn_units: [[1024]] dnn_units: [[1024, 1024, 1024, 1024]] regularizerScale: [0.1] dropoutDnn: [0.0,0.25,0.5,0.8] training_percentage: 80 filenameX: 257MX.csv	Saving weights with lr=0.001, batchsize=512, seq_length=20, dnn=[1024, 1024, 1024, 1024], rnn=[1024], iter=4076, test loss=0.5008381605148315, dr=0.0, re=0.1	Results----------- Total L2 Test Loss: 2.637200533723681 Total POSTSCALED MAE: 0.4110878863110178
D4	bigNoYRE20201102-131954_log.txt	NoYScaling, rnn activ = sigmoid dnn activ = relu, standardScale train loss= L1, optimizer = Adam, kernel_L2_regularizer	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [20] learning_rates: [0.001] rnn_units: [[2048]] dnn_units: [[4092]] regularizerScale: [0.1] dropoutDnn: [0.0,0.25,0.5,0.8] training_percentage: 80 filenameX: 257MX.csv	Saving weights with lr=0.001, batchsize=512, seq_length=20, dnn=[4092], rnn=[2048], iter=2921, test loss=0.45967569947242737, dr=0.0, re=0.1	Results----------- Total L2 Test Loss: 6.879297393659841 Total POSTSCALED MAE: 0.9437224947608597

Figure B.13.: LogsD01-04



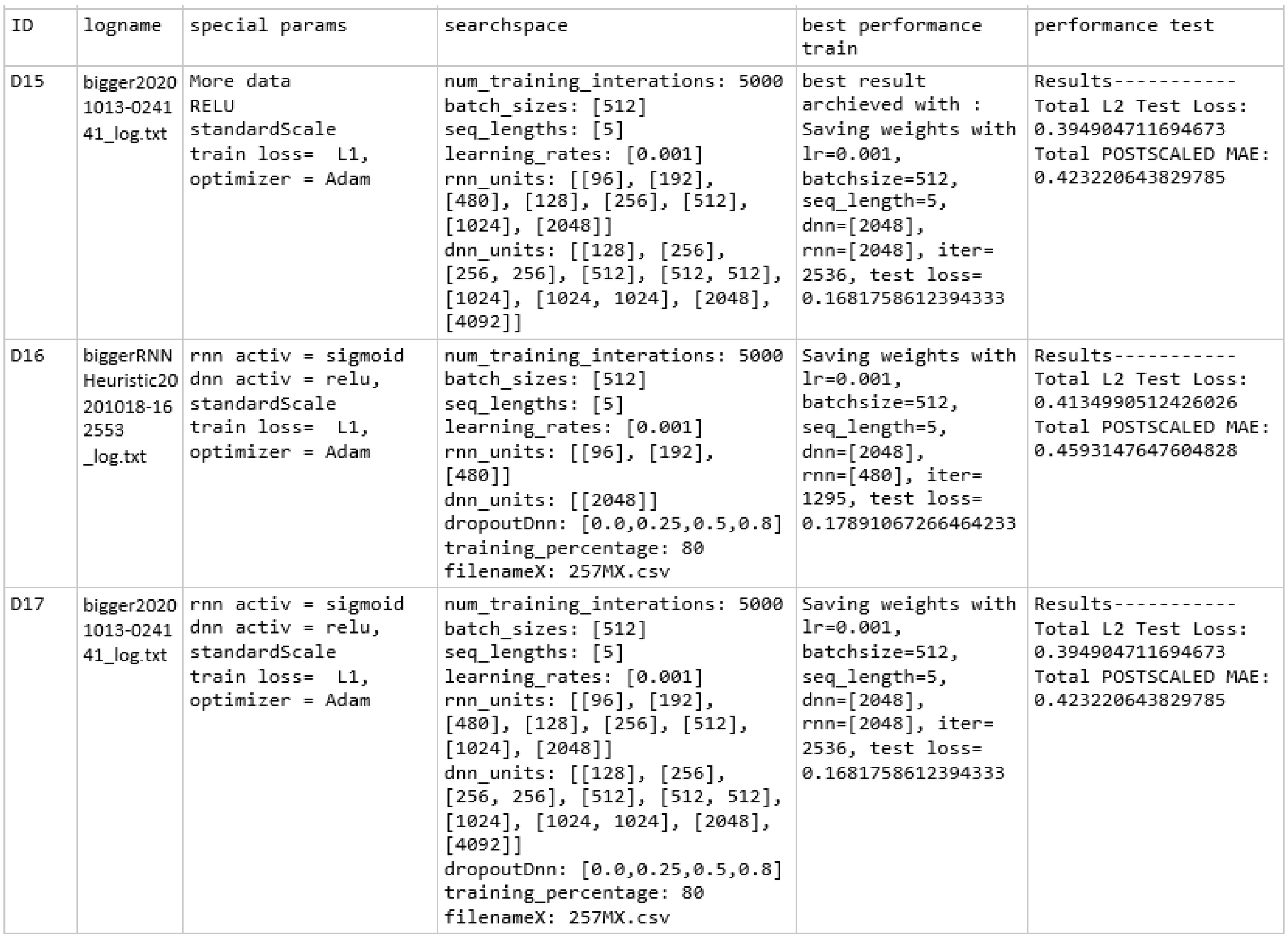
ID	logname	special params	searchspace	best performance train	performance test
D5	bigtestBest WithREDR 20201028-155431 _log.txt	rnn activ = sigmoid dnn activ = relu, standardScale train loss= L1, optimizer = Adam, kernel_L2_regularizer	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.001] rnn_units: [[2048]] dnn_units: [[4092]] cnn_units: [[None]] regularizerScale: [0.001, 0.01, 0.1, 0.5] dropoutDnn: [0.0,0.25,0.5,0.8] training_percentage: 80 filenameX: 257MX.csv	Saving weights with lr=0.001, batchsize=512, seq_length=5, dnn=[4092], rnn=[2048], iter= 3476, test loss= 0.15379643440246582 , dr=0.25, re=0.1	Results----------- Total L2 Test Loss: 0.4188848036678444 Total POSTSCALED MAE: 0.4203603198642186
D6	bigRnn96d More2020 1025-1747 25_log.txt	rnn activ = sigmoid dnn activ = relu, standardScale train loss= L1, optimizer = Adam,	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [5, 10, 20] learning_rates: [0.001] rnn_units: [[96], [96, 96], [96, 96, 96], [96, 96, 96, 96, 96], [96, 96, 96, 96, 96, 96], [96, 96, 96, 96]] dnn_units: [[2048]] cnn_units: [[None]] training_percentage: 80 filenameX: 257MX.csv	Saving weights with lr=0.001, batchsize=512, seq_length=10, dnn=[2048], rnn=[96], iter= 1762, test loss= 0.18969134986400604	Results----------- Total L2 Test Loss: 0.5338730397738185 Total POSTSCALED MAE: 0.5248275496895339
D7	bigRnn96C nn3202010 20-114210 _log.txt	rnn activ = sigmoid dnn activ = relu, standardScale train loss= L1, optimizer = Adam,	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [8] learning_rates: [0.001] rnn_units: [[96]] dnn_units: [[2048]] cnn_units: [[96]] training_percentage: 80 filenameX: 257MX.csv	Saving weights with lr=0.001, batchsize=512, seq_length=8, dnn=[2048], rnn=[96], iter= 1342, test loss= 0.20823879539966583	Results----------- Total L2 Test Loss: 1.299111843985979 Total POSTSCALED MAE: 0.8291683298712877
D8	bigRnn962 0201020-1 04054 _log.txt	rnn activ = sigmoid dnn activ = relu, standardScale train loss= L1, optimizer = Adam,	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [5, 8, 10, 15] learning_rates: [0.001] rnn_units: [[96]] dnn_units: [[1024], [2048], [4092]] cnn_units: [[None]] training_percentage: 80 filenameX: 257MX.csv	Saving weights with lr=0.001, batchsize=512, seq_length=8, dnn=[2048], rnn=[96], iter= 3686, test loss= 0.18442770838737488	Results----------- Total L2 Test Loss: 0.5249747056984038 Total POSTSCALED MAE: 0.5305078092989963
D9	bigRnnTest 20201020-032922 _log.txt	rnn activ = sigmoid dnn activ = relu, standardScale train loss= L1, optimizer = Adam,	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.001] rnn_units: [[4092]] dnn_units: [[4092]] cnn_units: [[None]] training_percentage: 80 filenameX: 257MX.csv	Saving weights with lr=0.001, batchsize=512, seq_length=5, dnn=[4092], rnn=[4092], iter= 1346, test loss= 0.16695702075958252	Results----------- Total L2 Test Loss: 0.4399953303783214 Total POSTSCALED MAE: 0.4409432417919804

Figure B.14.: LogsD05-09



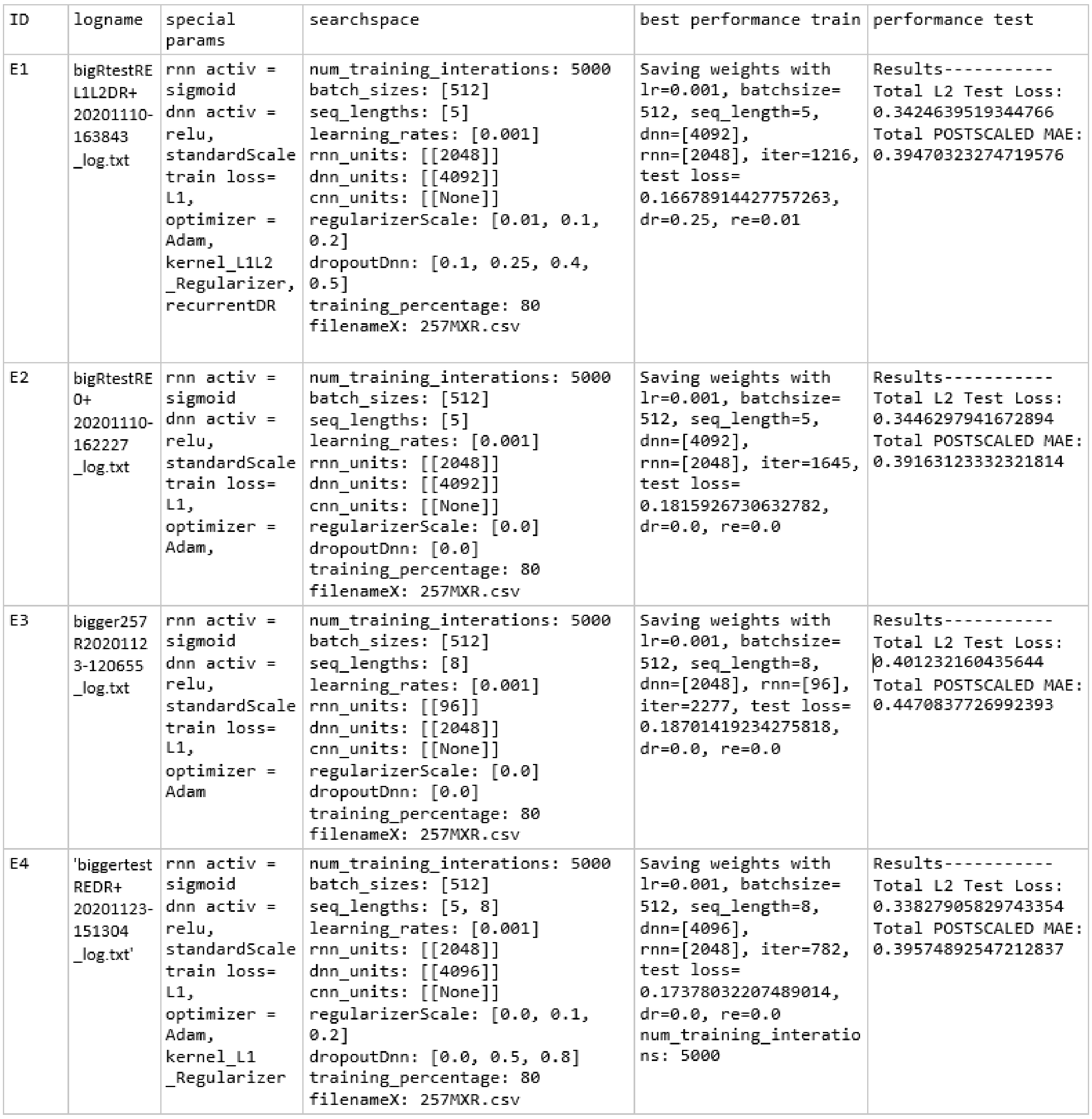
ID	logname	special params	searchspace	best performance train	performance test
D10	bigCnnTemporal20201019-162738_log.txt	rnn activ = sigmoid dnn activ = relu, standardScale train loss= L1, optimizer = Adam,	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.001] rnn_units: [[128], [256], [512]] dnn_units: [[1024], [2048], [4092]] cnn_units: [[256], [512], [1024]] training_percentage: 80 filenameX: 257MX.csv	best result archieved with : Saving weights with lr=0.001, batchsize=512, seq_length=5, dnn=[4092], rnn=[512], iter= 2179, test loss= 0.21534764766693115	Results----------- Total L2 Test Loss: 0.678151327523417 Total POSTSCALED MAE: 0.5745845215948623
D11	bigCnnTemporal20201019-162738_log.txt	rnn activ = sigmoid dnn activ = relu, standardScale train loss= L1, optimizer = Adam	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.001] rnn_units: [[128], [256], [512]] dnn_units: [[1024], [2048], [4092]] cnn_units: [[256], [512], [1024]] training_percentage: 80 filenameX: 257MX.csv	Saving weights with lr=0.001, batchsize=512, seq_length=5, dnn=[4092], rnn=[512], iter= 2179, test loss= 0.21534764766693115	Results----------- Total L2 Test Loss: 0.678151327523417 Total POSTSCALED MAE: 0.5745845215948623
D12	bigCnnTemporalNoPool20201019-172926_log.txt	rnn activ = sigmoid dnn activ = relu, standardScale train loss= L1, optimizer = Adam noPooling	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.001] rnn_units: [[128], [256], [512]] dnn_units: [[1024], [2048], [4092]] cnn_units: [[256], [512], [1024]] training_percentage: 80 filenameX: 257MX.csv	Saving weights with lr=0.001, batchsize=512, seq_length=5, dnn=[4092], rnn=[512], iter= 748, test loss= 0.1828904002904892	Results----------- Total L2 Test Loss: 0.7968247758314113 Total POSTSCALED MAE: 0.6763278870401255
D13	bigRnnTest20201020-032922_log.txt	rnn activ = sigmoid dnn activ = relu, standardScale train loss= L1, optimizer = Adam	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.001] rnn_units: [[4092]] dnn_units: [[4092]] training_percentage: 80 filenameX: 257MX.csv	Saving weights with lr=0.001, batchsize=512, seq_length=5, dnn=[4092], rnn=[4092], iter= 1346, test loss= 0.16695702075958252	Results----------- Total L2 Test Loss: 0.4399953303783214 Total POSTSCALED MAE: 0.4409432417919804
D14	bigger20201013-005700_log.txt	rnn activ = sigmoid dnn activ = relu, standardScale train loss= L1, optimizer = Adam	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.001] rnn_units: [[2048]] dnn_units: [[4092]] training_percentage: 80 filenameX: 257MX.csv	Saving weights with lr=0.001, batchsize=512, seq_length=5, dnn=[4092], rnn=[2048], iter= 1811, test loss= 0.1629171371459961	Results----------- Total L2 Test Loss: 0.3782937365497573 Total POSTSCALED MAE: 0.41305353132640377

Figure B.15.: LogsD10-14



ID	logname	special params	searchspace	best performance train	performance test
D15	bigger20201013-024141_log.txt	More data RELU standardScale train loss= L1, optimizer = Adam	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.001] rnn_units: [[96], [192], [480], [128], [256], [512], [1024], [2048]] dnn_units: [[128], [256], [256, 256], [512], [512, 512], [1024], [1024, 1024], [2048], [4092]]	best result archieved with : Saving weights with lr=0.001, batchsize=512, seq_length=5, dnn=[2048], rnn=[2048], iter= 2536, test loss= 0.1681758612394333	Results----------- Total L2 Test Loss: 0.394904711694673 Total POSTSCALED MAE: 0.423220643829785
D16	biggerRNNHeuristic20201018-162553_log.txt	rnn activ = sigmoid dnn activ = relu, standardScale train loss= L1, optimizer = Adam	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.001] rnn_units: [[96], [192], [480]] dnn_units: [[2048]] dropoutDnn: [0.0,0.25,0.5,0.8] training_percentage: 80 filenameX: 257MX.csv	Saving weights with lr=0.001, batchsize=512, seq_length=5, dnn=[2048], rnn=[480], iter= 1295, test loss= 0.17891067266464233	Results----------- Total L2 Test Loss: 0.4134990512426026 Total POSTSCALED MAE: 0.4593147647604828
D17	bigger20201013-024141_log.txt	rnn activ = sigmoid dnn activ = relu, standardScale train loss= L1, optimizer = Adam	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.001] rnn_units: [[96], [192], [480], [128], [256], [512], [1024], [2048]] dnn_units: [[128], [256], [256, 256], [512], [512, 512], [1024], [1024, 1024], [2048], [4092]] dropoutDnn: [0.0,0.25,0.5,0.8] training_percentage: 80 filenameX: 257MX.csv	Saving weights with lr=0.001, batchsize=512, seq_length=5, dnn=[2048], rnn=[2048], iter= 2536, test loss= 0.1681758612394333	Results----------- Total L2 Test Loss: 0.394904711694673 Total POSTSCALED MAE: 0.423220643829785

Figure B.16.: LogsD15-17



ID	logname	special params	searchspace	best performance train	performance test
E1	bigRtestREL1L2DR+20201110-163843_log.txt	rnn activ = sigmoid dnn activ = relu, standardScale train loss= L1, optimizer = Adam, kernel_L1L2_Regularizer, recurrentDR	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.001] rnn_units: [[2048]] dnn_units: [[4092]] cnn_units: [[None]] regularizerScale: [0.01, 0.1, 0.2] dropoutDnn: [0.1, 0.25, 0.4, 0.5] training_percentage: 80 filenameX: 257MXR.csv	Saving weights with lr=0.001, batchsize= 512, seq_length=5, dnn=[4092], rnn=[2048], iter=1216, test loss= 0.16678914427757263, dr=0.25, re=0.01	Results----------- Total L2 Test Loss: 0.3424639519344766 Total POSTSCALED MAE: 0.39470323274719576
E2	bigRtestRE0+20201110-162227_log.txt	rnn activ = sigmoid dnn activ = relu, standardScale train loss= L1, optimizer = Adam,	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [5] learning_rates: [0.001] rnn_units: [[2048]] dnn_units: [[4092]] cnn_units: [[None]] regularizerScale: [0.0] dropoutDnn: [0.0] training_percentage: 80 filenameX: 257MXR.csv	Saving weights with lr=0.001, batchsize= 512, seq_length=5, dnn=[4092], rnn=[2048], iter=1645, test loss= 0.1815926730632782, dr=0.0, re=0.0	Results----------- Total L2 Test Loss: 0.3446297941672894 Total POSTSCALED MAE: 0.39163123332321814
E3	bigger257R20201123-120655_log.txt	rnn activ = sigmoid dnn activ = relu, standardScale train loss= L1, optimizer = Adam	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [8] learning_rates: [0.001] rnn_units: [[96]] dnn_units: [[2048]] cnn_units: [[None]] regularizerScale: [0.0] dropoutDnn: [0.0] training_percentage: 80 filenameX: 257MXR.csv	Saving weights with lr=0.001, batchsize= 512, seq_length=8, dnn=[2048], rnn=[96], iter=2277, test loss= 0.18701419234275818, dr=0.0, re=0.0	Results----------- Total L2 Test Loss: 0.401232160435644 Total POSTSCALED MAE: 0.4470837726992393
E4	'biggertestREDR+20201123-151304_log.txt'	rnn activ = sigmoid dnn activ = relu, standardScale train loss= L1, optimizer = Adam, kernel_L1_Regularizer	num_training_interations: 5000 batch_sizes: [512] seq_lengths: [5, 8] learning_rates: [0.001] rnn_units: [[2048]] dnn_units: [[4096]] cnn_units: [[None]] regularizerScale: [0.0, 0.1, 0.2] dropoutDnn: [0.0, 0.5, 0.8] training_percentage: 80 filenameX: 257MXR.csv	Saving weights with lr=0.001, batchsize= 512, seq_length=8, dnn=[4096], rnn=[2048], iter=782, test loss= 0.17378032207489014, dr=0.0, re=0.0 num_training_interations: 5000	Results----------- Total L2 Test Loss: 0.33827905829743354 Total POSTSCALED MAE: 0.39574892547212837

Figure B.17.: LogsE01-04

C. Code Appendix

C.1. Tensorflow Server Scaling

Listed are the implementations for MinMax, Standard and Featurewise standard Scaling. Input data comes in as float array with sequence length * number of bones * six length. prediction data comes as float array with number of bones * six length. LayerInputShape is (-1, sequence_length, number of bones * 6)

C.1.1. MinMax Scaling

```
 MinMaxScaleDataInput(input)
{
        var inputArr = Array.from(input);
        const out = inputArr.map( (element,index) =>

                (element - this.scalerXMin[index % this.layerInputShape[2]]) /
                (this.scalerXMax[index % this.layerInputShape[2]]
                 - this.scalerXMin[index % this.layerInputShape[2]])
        );
        return out;
}

MinMaxScalePrediction(prediction)
{
        var inputArr = Array.from(prediction);
        var out = inputArr.map( (element,index) =>

                element * (this.scalerYMax[index % this.layerInputShape[2] ] -
                 this.scalerYMin[index % this.layerInputShape[2]]) +
                 this.scalerYMin[index % this.layerInputShape[2]]
        );
        return out; }
```

C.1.2. Standard Scaling

```
standardScaleDataInput(input)
{
        var inputArr = Array.from(input);
        const out = inputArr.map( (element,index) =>

                (element - this.meanX[index % this.layerInputShape[2]]) /
                this.scaleX[index % this.layerInputShape[2]]
        );
        return out;
}

standardScalePrediction(prediction)
{
var inputArr = Array.from(prediction);
const out = inputArr.map( (element,index) =>

element * this.scaleY[index % this.layerInputShape[2]]
+ this.meanY[index % this.layerInputShape[2]]
);
console.log(out);
return out;
}
```

C.1.3. Feature Scaling

```
 standardFeatureScaleDataInput(input)
{
	var inputArr = Array.from(input);
	const out = inputArr.map( (element,index) =>

		(element - this.meanX[(index % 6)]) / this.scaleX[(index % 6)]
	);
	return out;
}
standardFeatureScalePrediction(prediction)
{
	var inputArr = Array.from(prediction);
	const out = inputArr.map( (element,index) =>

		element * this.scaleY[index % 6] + this.meanY[index % 6 ]
	);
	return out;
}
```

List of Figures

Bibliography

[Ado] Adobe. URL: `https://www.mixamo.com/` (visited on 11/17/2020).

[AF09] B. Allen and P. Faloutsos. "Complex networks of simple neurons for bipedal locomotion." In: *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2009, pp. 4457–4462.

[Ara10] M. Araki. "Control Systems, Robotics and Automation—vol II—PID Control." In: *Kyoto University, Japan* (2010).

[AS20] A. Amini and A. Soleimany. *MIT 6.S191: Introduction to Deep Learning*. Jan. 2020. URL: `http://introtodeeplearning.com/` (visited on 05/12/2020).

[AWA17] V. Aparanji, U. V. Wali, and R. Aparna. "Robotic motion control using machine learning techniques." In: *2017 International Conference on Communication and Signal Processing (ICCSP)*. IEEE. 2017, pp. 1241–1245.

[BC03] G. C. Burdea and P. Coiffet. *Virtual reality technology*. John Wiley & Sons, 2003.

[Bot+00] C. Botella, R. M. Baños, H. Villa, C. Perpiñá, and A. Garcıa-Palacios. "Virtual reality in the treatment of claustrophobic fear: A controlled, multiple-baseline design." In: *Behavior therapy* 31.3 (2000), pp. 583–595.

[Bou02a] D. M. Bourg. *Physics for game developers*. " O'Reilly Media, Inc.", 2002, p. 2.

[Bou02b] D. M. Bourg. *Physics for game developers*. " O'Reilly Media, Inc.", 2002, p. 69.

[Bro20a] J. Brownlee. *How to Use StandardScaler and MinMaxScaler Transforms in Python*. Aug. 2020. URL: `https://machinelearningmastery.com/standardscaler-and-minmaxscaler-transforms-in-python/` (visited on 11/28/2020).

[Bro20b] J. Brownlee. *Regression Tutorial with the Keras Deep Learning Library in Python*. Aug. 2020. URL: `https://machinelearningmastery.com/regression-tutorial-keras-deep-learning-library-python/` (visited on 11/22/2020).

[Cav+98] M. Cavazza, R. Earnshaw, N. Magnenat-Thalmann, and D. Thalmann. "Motion control of virtual humans." In: *IEEE Computer Graphics and Applications* 18.5 (1998), pp. 24–31.

[CN01] L. Chen and K. S. Narendra. "Nonlinear adaptive control using neural networks and multiple models." In: *Automatica* 37.8 (2001), pp. 1245–1255.

[Cor] N. Corporation. *Rigid Body Dynamics*. URL: https://docs.nvidia.com/gameworks/content/gameworkslibrary/physx/guide/Manual/RigidBodyDynamics.html (visited on 11/16/2020).

[Csá+01] B. C. Csáji et al. "Approximation with artificial neural networks." In: *Faculty of Sciences, Etvs Lornd University, Hungary* 24.48 (2001), p. 7.

[Dam12] H. Dammertz. *Hammersley Points on the Hemisphere*. 2012. URL: http://holger.dammertz.org/stuff/notes_HammersleyOnHemisphere.html (visited on 11/29/2020).

[Dam19] J. Damiani. *Virtual Market 3, The Largest SocialVR Convention In The World, Opens In VRChat*. Sept. 2019. URL: https://www.forbes.com/sites/jessedamiani/2019/09/23/virtual-market-3-the-largest-socialvr-convention-in-the-world-opens-in-vrchat/?sh=332c1ea83ca1 (visited on 11/11/2020).

[dev20a] scikit-learn developers. *sklearn.preprocessing.MinMaxScaler*. 2007 - 2020. URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html (visited on 11/30/2020).

[dev20b] scikit-learn developers. *sklearn.preprocessing.StandardScaler*. 2007 - 2020. URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html (visited on 11/30/2020).

[DHS11] J. Duchi, E. Hazan, and Y. Singer. "Adaptive subgradient methods for online learning and stochastic optimization." In: *Journal of machine learning research* 12.7 (2011).

[Die+15] J. Diemer, G. W. Alpers, H. M. Peperkorn, Y. Shiban, and A. Mühlberger. "The impact of perception and presence on emotional reactions: a review of research in virtual reality." In: *Frontiers in psychology* 6 (2015), p. 26.

[DLA20] P. Dubs, J. Z. Loh, and S. Azeem. *Early Stopping*. 2020. URL: https://deeplearning4j.konduit.ai/tuning-and-training/early-stopping (visited on 11/15/2020).

[Du+19] S. Du, J. Lee, H. Li, L. Wang, and X. Zhai. "Gradient descent finds global minima of deep neural networks." In: *International Conference on Machine Learning*. 2019, pp. 1675–1685.

[Fis+87] S. S. Fisher, M. McGreevy, J. Humphries, and W. Robinett. "Virtual environment display system." In: *Proceedings of the 1986 workshop on Interactive 3D graphics*. 1987, pp. 77–87.

[Fol87] J. D. Foley. "Interfaces for advanced computing." In: *Scientific American* 257.4 (1987), pp. 126–135.

[GJP95] F. Girosi, M. Jones, and T. Poggio. “Regularization theory and neural networks architectures.” In: *Neural computation* 7.2 (1995), pp. 219–269.

[Goo+16a] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*. Vol. 1. MIT press Cambridge, 2016, p. 168.

[Goo+16b] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*. Vol. 1. MIT press Cambridge, 2016, pp. 285–286.

[Goo+16c] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*. Vol. 1. MIT press Cambridge, 2016, p. 231.

[Goo+16d] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*. Vol. 1. MIT press Cambridge, 2016, p. 227.

[Goo+16e] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*. Vol. 1. MIT press Cambridge, 2016, p. 243.

[Goo+16f] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*. Vol. 1. MIT press Cambridge, 2016, p. 398.

[Gro+04] K. Grochow, S. L. Martin, A. Hertzmann, and Z. Popović. “Style-based inverse kinematics.” In: *ACM SIGGRAPH 2004 Papers*. 2004, pp. 522–531.

[GTH98] R. Grzeszczuk, D. Terzopoulos, and G. Hinton. “Neuroanimator: Fast neural network emulation and control of physics-based models.” In: *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. 1998, pp. 9–20.

[Haw04] D. M. Hawkins. “The problem of overfitting.” In: *Journal of chemical information and computer sciences* 44.1 (2004), pp. 1–12.

[HNP09] A. Halevy, P. Norvig, and F. Pereira. “The unreasonable effectiveness of data.” In: *IEEE Intelligent Systems* 24.2 (2009), pp. 8–12.

[Hoc98] S. Hochreiter. “The vanishing gradient problem during learning recurrent neural nets and problem solutions.” In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 6.02 (1998), pp. 107–116.

[HS97] S. Hochreiter and J. Schmidhuber. “Long short-term memory.” In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[IM98] P. Indyk and R. Motwani. “Approximate nearest neighbors: towards removing the curse of dimensionality.” In: *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. 1998, pp. 604–613.

[IS15] S. Ioffe and C. Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift.” In: *arXiv preprint arXiv:1502.03167* (2015).

[Jad+14] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman. "Synthetic data and artificial neural networks for natural scene text recognition." In: *arXiv preprint arXiv:1406.2227* (2014).

[JKK20] A. D. Jagtap, K. Kawaguchi, and G. E. Karniadakis. "Adaptive activation functions accelerate convergence in deep and physics-informed neural networks." In: *Journal of Computational Physics* 404 (2020), p. 109136.

[Kan+13] M. R. Kandalaft, N. Didehbani, D. C. Krawczyk, T. T. Allen, and S. B. Chapman. "Virtual reality social cognition training for young adults with high-functioning autism." In: *Journal of autism and developmental disorders* 43.1 (2013), pp. 34–44.

[Kar20] K. Karas. "VR Re-Embodiment: Establishing a Control Structure to enable Physic-based Movement of the Human Body in Unity 3D." Bachelor's Thesis in Informatics: Games Engineering. TUM, Apr. 15, 2020.

[Kaw90] M. Kawato. "Feedback-error-learning neural network for supervised motor learning." In: *Advanced neural computers*. Elsevier, 1990, pp. 365–372.

[KB14] D. P. Kingma and J. Ba. "Adam: A method for stochastic optimization." In: *arXiv preprint arXiv:1412.6980* (2014).

[Ker] Keras. *LSTM layer*. URL: `https://keras.io/api/layers/recurrent_layers/lstm/` (visited on 11/10/2020).

[KH10] A. Krizhevsky and G. Hinton. "Convolutional deep belief networks on cifar-10." In: *Unpublished manuscript* 40.7 (2010), pp. 1–9.

[LF03] A. Linden and J. Fenn. "Understanding Gartner's hype cycles." In: *Strategic Analysis Report Nº R-20-1971. Gartner, Inc* (2003), p. 88.

[Mar+17] J. Martın-Gutiérrez, C. E. Mora, B. Añorbe-Dıaz, and A. González-Marrero. "Virtual technologies trends in education." In: *EURASIA Journal of Mathematics, Science and Technology Education* 13.2 (2017), pp. 469–486.

[MFS07] M. Mittmann, J. Francik, and A. Szarowicz. "Physics-based animation of human avatars." PhD thesis. Citeseer, 2007.

[Miy+88] H. Miyamoto, M. Kawato, T. Setoyama, and R. Suzuki. "Feedback-error-learning neural network for trajectory control of a robotic manipulator." In: *Neural networks* 1.3 (1988), pp. 251–265.

[MP16] H. N. Mhaskar and T. Poggio. "Deep vs. shallow networks: An approximation theory perspective." In: *Analysis and Applications* 14.06 (2016), pp. 829–848.

[Net] NetMQ. *NetMQ*. URL: `https://netmq.readthedocs.io/en/latest/` (visited on 11/10/2020).

[NK] A. NG and K. Katanforoosh. *Hyperparameter Tuning and Tensorboard*. URL: `https://cs230.stanford.edu/section/9/` (visited on 11/15/2020).

[Ooe+13] R. Ooe, I. Suzuki, M. Yamamoto, and M. Furukawa. "Study on evolution of the artificial flying creature controlled by neuro-evolution." In: *Artificial Life and Robotics* 17.3-4 (2013), pp. 470–475.

[PA07] M. Price and P. Anderson. "The role of presence in virtual reality exposure therapy." In: *Journal of anxiety disorders* 21.5 (2007), pp. 742–751.

[Pad+16] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli. "A survey of motion planning and control techniques for self-driving urban vehicles." In: *IEEE Transactions on intelligent vehicles* 1.1 (2016), pp. 33–55.

[PTK87] T. Poggio, V. Torre, and C. Koch. "Computational vision and regularization theory." In: *Readings in computer vision* (1987), pp. 638–643.

[Res19] G. V. Research. *Virtual Reality Headset Market Size, Share & Trends Analysis Report By End Device (Low-end, Mid-range, High-end), By Product Type, By Application Type (Gaming, Education), By Region, And Segments Forecasts, 2019 - 2025*. Nov. 2019. URL: `https://www.grandviewresearch.com/industry-analysis/virtual-reality-vr-headset-market` (visited on 10/05/2020).

[RH02] T. Reil and P. Husbands. "Evolution of central pattern generators for bipedal walking in a real-time physics environment." In: *IEEE Transactions on Evolutionary Computation* 6.2 (2002), pp. 159–168.

[RM12] G. Riva and F. Mantovani. "Being there: Understanding the feeling of presence in a synthetic environment and its potential for clinical change." In: *Virtual reality in psychological, medical and pedagogical applications* (2012), pp. 3–34.

[RN16a] S. J. Russell and P. Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited, 2016, p. 728.

[RN16b] S. J. Russell and P. Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited, 2016, p. 719.

[Rot+00] B. O. Rothbaum, L. Hodges, S. Smith, J. H. Lee, and L. Price. "A controlled study of virtual reality exposure therapy for the fear of flying." In: *Journal of consulting and Clinical Psychology* 68.6 (2000), p. 1020.

[Rot+99] B. O. Rothbaum, L. Hodges, R. Alarcon, D. Ready, F. Shahar, K. Graap, J. Pair, P. Hebert, D. Gotz, B. Wills, et al. "Virtual reality exposure therapy for PTSD Vietnam veterans: A case study." In: *Journal of Traumatic Stress: Official Publication of The International Society for Traumatic Stress Studies* 12.2 (1999), pp. 263–271.

[San20] W. Sandro. *Ubi-Interact Overview*. July 2020. URL: `https://wiki.tum.de/display/infar/Ubi-Interact+Overview` (visited on 10/10/2020).

[SBR12] C. Stanton, A. Bogdanovych, and E. Ratanasena. "Teleoperation of a humanoid robot using full-body motion capture, example movements, and machine learning." In: *Proc. Australasian Conference on Robotics and Automation*. Vol. 8. 2012, p. 51.

[SD91] J. Sietsma and R. J. Dow. "Creating artificial neural networks that generalize." In: *Neural networks* 4.1 (1991), pp. 67–79.

[Set97] R. Setiono. "A penalty-function approach for pruning feedforward neural networks." In: *Neural computation* 9.1 (1997), pp. 185–204.

[She92] T. B. Sheridan. "Musings on telepresence and virtual presence." In: *Presence: Teleoperators & Virtual Environments* 1.1 (1992), pp. 120–126.

[Sla09] M. Slater. "Place illusion and plausibility can lead to realistic behaviour in immersive virtual environments." In: *Philosophical Transactions of the Royal Society B: Biological Sciences* 364.1535 (2009), pp. 3549–3557.

[Sri+14] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting." In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[SSC10] M. Slater, B. Spanlang, and D. Corominas. "Simulating virtual environments within virtual environments as the basis for a psychophysics of presence." In: *ACM Transactions on Graphics (TOG)* 29.4 (2010), pp. 1–9.

[Sta16] Stanford. *AdaGrad - Adaptive Subgradient Methods*. July 2016. URL: `https://ppasupat.github.io/a9online/1107.html` (visited on 11/11/2020).

[Sta20] N. Statt. *Four Seasons Total Landscaping becomes a VRChat hangout for furries*. Nov. 2020. URL: `https://www.theverge.com/tldr/2020/11/9/21557029/four-seasons-total-landscaping-furries-vrchat-trump` (visited on 11/11/2020).

[Ste20a] Steamcharts. *Half Live Alyx*. Oct. 2020. URL: `https://steamcharts.com/app/438100` (visited on 10/25/2020).

[Ste20b] Steamcharts. *VRChat*. Oct. 2020. URL: `https://steamcharts.com/app/438100` (visited on 10/24/2020).

[SU93] M. Slater and M. Usoh. “Representations systems, perceptual position, and presence in immersive virtual environments.” In: *Presence: Teleoperators & Virtual Environments* 2.3 (1993), pp. 221–233.

[Sut68] I. E. Sutherland. “A head-mounted three dimensional display.” In: *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*. 1968, pp. 757–764.

[Tan20] H. Tankovska. *Augmented and virtual reality (AR/VR) headset shipments worldwide from 2020 to 2025*. Aug. 2020. URL: `https://www.statista.com/statistics/653390/worldwide-virtual-and-augmented-reality-headset-shipments/` (visited on 10/05/2020).

[Tec] 2. U. Technologies. *Physics*. URL: `https://docs.unity3d.com/Manual/PhysicsSection.html` (visited on 11/10/2020).

[Tec20a] 2. U. Technologies. *Importing humanoid animations*. Oct. 2020. URL: `https://docs.unity3d.com/Manual/ConfiguringtheAvatar.html` (visited on 11/14/2020).

[Tec20b] 2. U. Technologies. *Rigidbody.AddForce*. Oct. 2020. URL: `https://docs.unity3d.com/ScriptReference/Rigidbody.AddForce.html` (visited on 11/28/2020).

[Tec20c] 2. U. Technologies. *Rigidbody.AddTorque*. Oct. 2020. URL: `https://docs.unity3d.com/ScriptReference/Rigidbody.AddTorque.html` (visited on 11/28/2020).

[Tec20d] 2. U. Technologies. *Rigidbody.MovePosition*. Oct. 2020. URL: `https://docs.unity3d.com/ScriptReference/Rigidbody.MovePosition.html` (visited on 11/28/2020).

[Tec20e] 2. U. Technologies. *Rigidbody.MoveRotation*. Oct. 2020. URL: `https://docs.unity3d.com/ScriptReference/Rigidbody.MoveRotation.html` (visited on 11/28/2020).

[Ten20] Tensorflow. *tf.keras.optimizers.Adam*. Oct. 2020. URL: `https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam` (visited on 11/09/2020).

[Viv20a] H. Vive. *Full-Body Tracking*. Sept. 2020. URL: `https://docs.vrchat.com/docs/full-body-tracking` (visited on 10/24/2020).

[Viv20b] H. Vive. *VIVE Tracker*. 2011 - 2020. URL: `https://www.vive.com/de/accessory/vive-tracker/` (visited on 10/24/2020).

[VOL08] J. Velagic, N. Osmic, and B. Lacevic. “Neural network controller for mobile robot motion control.” In: *World Academy of Science, Engineering and Technology* 47 (2008), pp. 193–198.

[Wil20] M. Wilson. *Valve Index stock quickly sells out, new orders will ship in up to 10 weeks*. Mar. 2020. URL: `https://www.kitguru.net/tech-news/featured-tech-news/matthew-wilson/valve-index-stock-quickly-sells-out-new-orders-will-ship-in-up-to-10-weeks/` (visited on 10/01/2020).

[WK19] S. Weber and G. Klinker. "VR Re-Embodiment in the Neurorobotics Platform." In: *Mensch und Computer 2019-Workshopband* (2019).

[Xu+19] Z. Xu, S. Bi, K. Sunkavalli, S. Hadap, H. Su, and R. Ramamoorthi. "Deep view synthesis from sparse photometric images." In: *ACM Transactions on Graphics (TOG)* 38.4 (2019), pp. 1–13.

[Zha+18] H. Zhang, S. Starke, T. Komura, and J. Saito. "Mode-adaptive neural networks for quadruped motion control." In: *ACM Transactions on Graphics (TOG)* 37.4 (2018), pp. 1–11.

[ZSV14] W. Zaremba, I. Sutskever, and O. Vinyals. "Recurrent neural network regularization." In: *arXiv preprint arXiv:1409.2329* (2014).

[ZXL18] Y. Zhao, Y. Xiong, and D. Lin. "Trajectory convolution for action recognition." In: *Advances in neural information processing systems*. 2018, pp. 2204–2215.