Embodied Perception during Walking using Deep Recurrent Neural Networks

by

Jacob Chen

B.Sc Electrical Engineering, University of Maryland, College Park, 2014

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES
(Computer Science)

The University of British Columbia
(Vancouver)

July 2017

© Jacob Chen, 2017

Abstract

Movements such as walking require knowledge of the environment in order to be robust. This knowledge can be gleaned via embodied perception. While information about the upcoming terrain such as compliance, friction, or slope may be difficult to directly estimate, using the walking motion itself allows for these properties to be implicitly observed over time from the stream of movement data. However, the relationship between a parameter such as ground compliance and the movement data may be complex and difficult to discover. In this thesis, we demonstrate the use of a Deep LSTM Network to estimate slope and ground compliance of terrain by observing a stream of sensory information that includes the character state and foot pressure information.

Lay Summary

Accurately estimating the environment that an agent is in is beneficial because it allows for immediate, robust, and effective responsiveness. Allowing the agent to build their own structured representations from input streams is a notion borrowed from the recent ideas in the cognitive sciences. In order to do this data driven approach, we use a modern machine learning model called the Deep Recurrent Neural Network, which enables the system to receive a stream of inputs that include the state of the bipedal agent as well as foot pressure sensors to estimate terrain properties that are difficult to analyze analytically. Using such a model allows the agent to build its own structured representations of the data that enable high fidelity estimation that may be used as supplemental information to inform decisions that the agent may make in the future.

Preface

My supervisor Michiel Van de Panne, was instrumental in providing the guidance and direction for this thesis. I was responsible for its implementation, experimental setup, and results, as well as the writing of this thesis.

Table of Contents

List of Tables

List of Figures

Glossary

SIMBICON SIMple BIped CONtrol

PD Proportional Derivative

LSTM Long Short Term Memory

RNN Recurrent Neural Network

BPTT Back Propagation Through Time

SVM Support Vector Machines

UP Universal Control Policy

OSI On-line System Identification Model

RELU Rectified Linear Unit

CEC Constant Error Carousel

EKF Extended Kalman Filters

POMDP Partially Observed Markhov Decision Processes

FFNN Feed Forward Neural Network

FSM Finite State Machine

COM Center Of Mass

Acknowledgments

I thank my supervisor Dr. Michiel van de Panne for his mentoring and support throughout my Master's track. I thoroughly appreciate your oversight, patience, and your willingness to work alongside your students. It is clearly evident that you take your students' interests to heart by the kinds of relationships you foster with them. Your inquisitive nature and character will be one that I will always strive to emulate.

I also thank my lab mates, Xue Bin Peng, Glen Berseth, and Shailen Agrawal that I had the pleasure and honor of meeting and sharing time in the lab. Even though I was not directly involved in your projects, you had no qualms in discussing your work. The enthusiasm you had for it was palpable and contagious. I was able to grow and learn so much from our discussions. I wish I could bother you guys forever. I couldn't have asked for a more fostering lab for knowledge.

I also extend my thanks to Dr. Dinesh Pai for being the second reader of my thesis.

Furthermore, I would like to thank Kimberly Dextras Romagnino for providing the necessary comedic relief of the lab. Without it, I may very well have gone insane.

Lastly, and most importantly, I thank my parents for all the support and love they have shown me. The opportunities that I have had would not have been possible without your encouragement, understanding, patience, and willingness to help. I would not be who I am today without your influences.

Chapter 1

Introduction

Our perception of the environment influences the way we interact with it. Thus, building an accurate perception that reflects reality accurately is crucial in informing the decisions we make. Furthermore, building an accurate representation of reality is restricted to the physical and sensory modalities that are afforded to the agent. The agent has the task of discovering the relationship mapping between information gleaned from its modalities and their environment. Traditional approaches for allowing an agent to discover this relationship require domain knowledge that is not representative of how humans and other organisms learn to discover and interact with their environment. For instance, in order to catch a ball, humans do not calculate the trajectory of the ball and predict where it will land. Instead, humans use the strategy of a constant stream of sensory information to position themselves in order to catch the ball successfully. In this work, we seek to emulate aspects of such behavior. This work seeks to explore insight related to sensory-motor embodiment and therefore how an agent should learn and interact with its environment without the need for embedding domain knowledge.

1.1 Motivations

A major theme in cognitive science is the idea of perception and its relation to cognition and how this is framed. Embodied perception and cognition is the idea that cognition is deeply dependent upon characteristics of the physical body of the

agent [Berthoz, 2002; Wilson and Foglia, 2017]. Part of the theory is that perception is tied to the physical make up of the agent and so the cognitive system of the agent is constrained by the capabilities of the physical modalities and movements afforded to the agent. The agent thus uses an interplay between the simulated consequences of its actions and the environment as a means of understanding their environment. This thesis addresses this theme by using the consequences of the environment on its own modalities through walking to understand and estimate terrain properties. By using the recent window of past consequences, we explore how this understanding can be achieved, possibly similar to how humans gain a better understanding through repeated temporal exposure and feedback. Thus this thesis' primary motivation is to explore and validate how well movement of an agent can help it understand its own environment and discover its relationship to latent variables that may be difficult to discover analytically.

Our work is also motivated by controls in animation and robotics. Typically in order to animate characters, motion capture techniques and/or artist involvement is required to provide realistic high quality animation. Physics based animation through control offers an alternative to these techniques along with many desirable benefits. If successful, these techniques for control provide responsive realism, interaction, and generalization towards a variety of environments. Using these kinds of approaches may reduce the effort and cost for animation by removing the need for tailoring animations to specific environments.

In the above context, this thesis also seeks to improve the capacity of using physics based controllers by introducing predictive models for environment state estimation. By using these models, the controllers gain the ability to anticipate changes and perturbations in the environment and respond intelligently. If the predictive model has enough fidelity, then it may be useful for the controller to augment the current state of the physical agent to propose an action. This work lies in the realm of state estimation and system identification where the agent uses an awareness of itself and its encompassing environment. Traditional system identification techniques include discovering the behaviors of a system through strategic sampling. Depending on the response of the system, the appropriate model is chosen and introduced to minimize the error between prediction and actual. Traditional state estimation techniques seek to estimate current state based on a combination of

new observations and previous estimate. This is most clearly demonstrated through the Kalman filter technique which is used extensively in many applications. Our work estimates the state of the environment through continuous sampling and constructs a model to approximate the system that transforms character movement and modality information to estimates of environmental latent variables, such as compliance and slope.

Deep Learning techniques have been shown to be successful in the field of computer graphics. Some examples include motion manifold learning and synthesis, physics based controllers, and 3D Mesh Labeling [Guo et al., 2015; Holden et al., 2016; Peng et al., 2016]. Additionally, much progress has been made using Recurrent Neural Network (RNN)s in Natural Language Processing due to its ability to process time sequenced inputs [Socher et al., 2013]. This work demonstrates the potential to leverage RNNs for the purpose of enhancing physics based controllers. Furthermore, deep learning is able to discover relationships between variables that may be difficult to discover analytically. This work relates environmental properties to the movement of the agent's afforded modalities which is difficult to model analytically. Here, the slope and compliance act as latent variables that affect the walking movement of the bipedal character. By using a data driven approach, we can discover this relationship with sufficient enough accuracy that there is no need to resort to complex analytical models.

We propose that by using the recent history of the states and other sensory information of a bipedal walker, we are able to find key patterns hidden in the state sequence that would enable the bipedal character to accurately estimate the ground properties of the terrain it is traversing such as slope and compliance. We thus allow the model to record and decide which elements in its input stream it should pay close attention to. An accurate estimate might only be obtainable during a particular part of the walk cycle, and so the model needs to remember this event in order to provide good estimates at other points in time. Similarly, the best result may only be obtainable by integrating together important pieces of information that are obtained at different points in the walk cycle. To accomplish this, we use a multi-layer LSTM network as our predictive model in order to learn temporal state dependencies that can be used for the prediction of ground properties. For this thesis, we require a walking controller and use the well known SIMBICON control

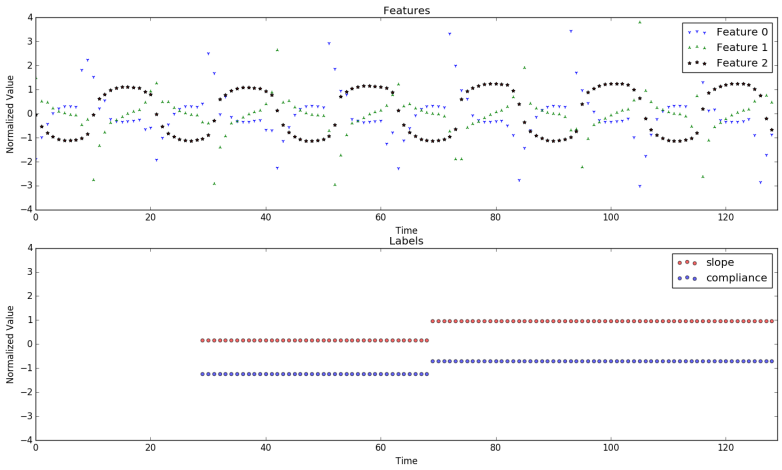
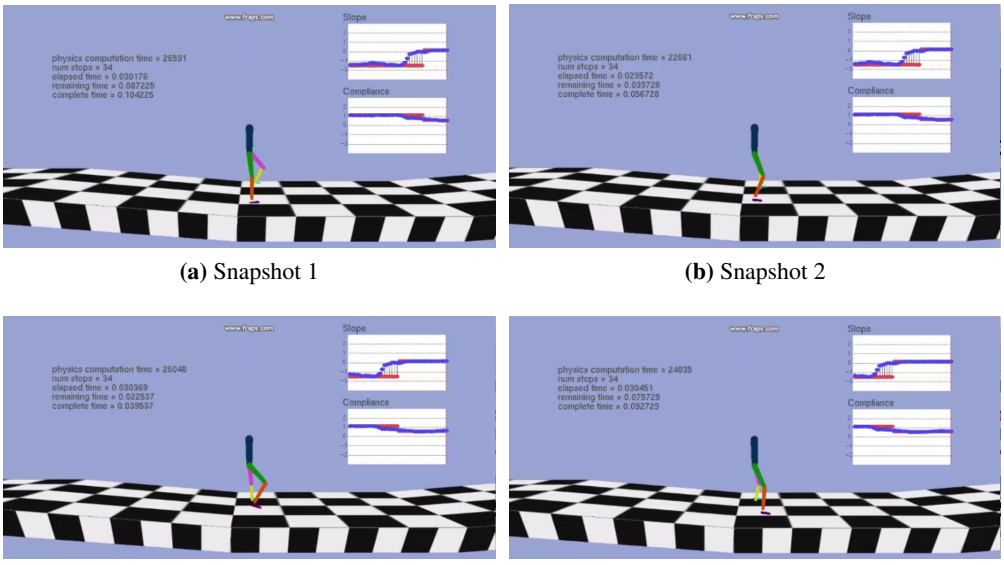


Figure 1.1: Sample features plotted with respect to time along with their labels

strategy [Yin et al., 2007] with a robust enough gait along with our own contact model in order to predict the slope and compliance of the terrain. This introduces the possibility to use high fidelity predictive models in order to enhance controls. The terrain follows a pattern of presentation, but each terrain segment consists of randomized slopes and ground compliance. We model ground compliance by using our own contact model based on springs and dampers. Further details on the simulation are found in Chapter 4. Figure 1.2 displays snapshots of the simulation. Figure 1.1 displays some state features plotted against time along with the terrain labels that the model uses to estimate terrain properties. The purpose is to illustrate the complex relationship that the model must learn to capture.

1.2 Thesis Overview

This thesis explores the effectiveness and potential of using deep Long Short Term Memory (LSTM) networks to build an embodied perception of the environment. In Chapter 2, we review the related work that has inspired this work. Chapter 3 serves to provide the background needed to understand the implementation of the thesis. We examine in fine detail each of the components that this thesis builds



(a) Snapshot 1 **(b)** Snapshot 2

(c) Snapshot 3 **(d)** Snapshot 4

Figure 1.2: Simulation snapshots in order from left to right, top to bottom. The character is estimating the terrain properties. Red circles represent the actual values, blue circles represent the predictions. All predictions are normalized with respect to the training set.

upon. Chapter 4 provides the details of the experimental methodology of the experiments and describes the simulation and strategy for obtaining our results. Chapter 5 presents and discusses the results of our experiments and compares them to provide meaningful conclusions. Lastly, Chapter 6 concludes with limitations of the work and the possible future directions it can take.

Chapter 2

Related Work

Using motion as a means for understanding the environment has its foundation in the cognitive sciences and extends into robotics and control. Furthermore, understanding the environment using time series data in a data driven fashion is still being explored as a major research topic. In this chapter, we provide the relevant work that shapes and defines our research question.

2.1 Embodied Perception

Embodied perception, also referred to as grounded perception, is the notion that perception and cognitive activity is rooted in sensorimotor experience, namely situated actions and bodily states [Barsalou, 2007]. This approach states that perception is shaped by the brain capturing the states and experiences across the modalities afforded to it and integrating them into a multi modal representation of an event. Later when the event is recalled, these representations are replayed as simulations that allow the brain to represent and reason across events. This type of approach shifts the paradigm of control strategies [Vernon, 2008]. Traditional strategies involve embedding symbols and intent into the agent and using those representations for learning and task completion. This strategy is known as the cognitivist position. The emergent strategy allows the agent to self organize and represent their environment given the modalities afforded it through agent-environment interaction over time. Recent advances in Machine Learning explore

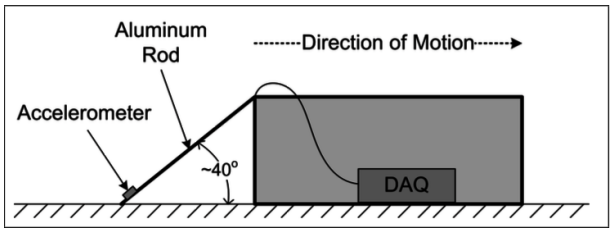


Figure 2.1: Simple contact sensor [Giguere and Dudek, 2009] used for surface identification

this paradigm by allowing representations of the data to emerge through repeated exposure and training instead of relying on hand-engineered symbols that are idealized descriptions of human cognitive representations.

2.2 Robot Locomotion with Environmental Knowledge

Locomotion in robotics has long been a challenging and continual effort. Creating robust controllers for robots requires exploiting the degrees of freedom inherent in the robot in a manner that optimizes its traversal across a dynamic environment. Thus, the more knowledge the robot has of itself and the environment the better it will perform. By decoupling system identification and control, each problem can be solved separately for more flexibility and robustness. System identification of the environment can be seen as separating the problem into Terrain Classification or Terrain Estimation.

2.2.1 Terrain Classification

A number of works have addressed terrain classification by leveraging machine learning techniques. A first approach is to determine the degree of simplification a modality can have to accomplish terrain classification. This approach uses a very simple contact dynamic sensor for surface identification as shown in Figure 2.1. Giguere and Dudek [2009] The authors attach an accelerometer to a rigid rod that contacts the surface and is dragged along by the robot to collect readings at a particular speed. The authors used a supervised learning method for surface identification of 10 surfaces and explore using an unsupervised technique for dis-

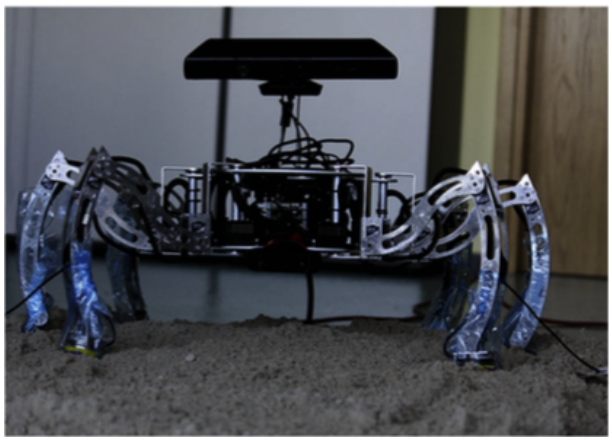


Figure 2.2: Legged Robot [Walas, 2015] used for classifying terrains

criminating against 2 surfaces. In the supervised setting, the data is aggregated with non-overlapping time-windows. Features are then extracted and processed by a 2 layer dense network. In this way, time dependence is preserved since sequential data is preprocessed before being presented to the network. The supervised setting achieved a score of 94.6%. In the unsupervised setting, the authors explore using a mixture of 2 Gaussian classifiers to classify the feature vector into 2 classes. The intuition is to discover the parameters that minimize the variability of classification across sequential data points. This setting was limited to the discriminatory power of the sensor across the 2 different terrains and was not competitive with their supervised approach but also was not fully explored. However, using unsupervised approaches is promising for terrain classification since it allows the agent to be trained online.

Another supervised approach is to classify 12 types of terrains by using a combination of visual, depth, and compliance data with a Support Vector Machines (SVM) classifier [Walas, 2015]. The data is collected and trained offline to find the best method for combining the separate classifiers from each sensing modality to maximize classification accuracy. Their best results come from combined classifiers from all three sensing modalities with a precision of 94.44% and a recall of 95.15%, which is sufficient to perform control actions based on this feedback. The authors modify the gait by changing the PD gains of their internal

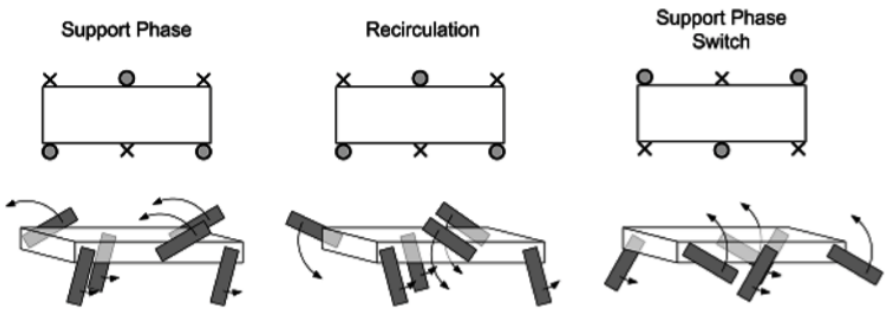


Figure 2.3: Robot [Sandeep Manjanna, 2013] used for classifying terrains using gait

controller. Depending on the type of terrain, the optimal gait is selected that will allow the robot to traverse the terrain in the most efficient manner with regards to speed and distance. The authors were able to see improvements using the optimal gait adjusted for terrain. The authors do not exploit any time dependencies in this work. The robot is displayed in Figure 2.2.

Finally an approach similar to our work is to identify the terrain using the gait itself. All sensory information is proprioceptive and does not acquire information about the surrounding environment [Sandeep Manjanna, 2013]. This work has ties to the emergent idea mentioned above of using movement and action to build perception. The authors found that classification of terrain was possible by analyzing the effect of different terrains on the gaits. The robot the authors experimented with is modeled in Figure 2.3. The robot was equipped with sensors measuring leg rotations, accelerations, rotations, magnetic orientation, as well as an estimation of electrical currents. The robot would then sample these modalities at a rate of 20Hz with the goal of classifying 4 types of terrain. They used an unsupervised clustering algorithm where each batch represents a sequence of consecutive sample measurements which embeds time dependent information. The parameters for the algorithm are discovered by attempting to make consistent classifications across consecutive temporal samples in the batch. The authors found that at particular settings of the robot, the data had more separability thereby making their classifier more effective. Using their unsupervised approach enabled them to achieve a success rate of 92.11% which is competitive with previous supervised approaches.

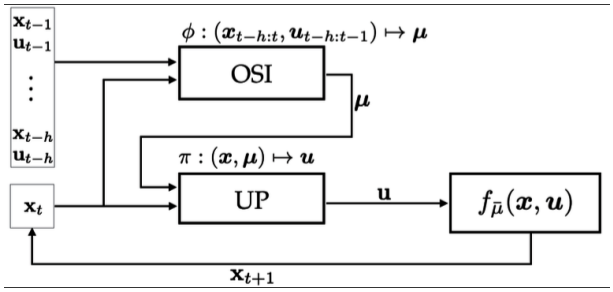


Figure 2.4: UP-OSI System Diagram of [Wenhao Yu, 2017]

It should be noted that the authors inject knowledge of optimal data separation in order for their algorithm to be the most effective rather than allowing that behavior to emerge automatically.

2.2.2 Terrain Estimation

Because of the many types of terrain, it may be beneficial to estimate the properties of the terrain rather than attempting to classify the terrain as being one of N discrete types. One work uses direct sensory information from the foot to analytically compute the surface gradient underneath the foot [Yi et al., 2010]. The agent follows a unique walking strategy to collect noisy sensory measurements of the pose and displacement of the foot at each step. This information is used to maintain and update a local estimate of the surface normal. The estimates are then used to adapt the gait. The learning is performed online for a small number of model parameters which allow learning to be achieved rapidly. The authors then compare using their method to enhance the locomotion of their robot and found it to be more stable and robust than using a baseline locomotion strategy.

Another approach to terrain estimation replaces LIDAR information with stereo vision to generate footstep plans for the robot on uneven terrain [Marion et al., 2015]. Using stereo vision, the authors build a 3D height map which then allows them to segment the terrain into possible regions for the footstep planning of robot. Using this approach, they found that the results were comparable to using LIDAR as a means for gathering the height map and can act as a direct replacement with the benefit of smaller weight and power costs.

In an attempt to leverage simulation to address a wide array of environmental conditions, the authors build a system utilizing neural networks which consists of a Universal Control Policy (UP) and a On-line System Identification Model (OSI) for terrain estimation [Wenhao Yu, 2017]. The UP outputs a control vector based on the current state and dynamic model parameters. The OSI uses a history of previous states and control vectors to predict the dynamic model parameters. The complete system diagram is shown in Figure 2.4. The intuition is that if the dynamic model parameters are correctly predicted for the environment, then the UP should be able to output the correct control vector that responds appropriately. The authors train the systems by sampling from a distribution of model parameters so that the system is robust to many variations of model parameters. During training, they couple the OSI model with the UP in order for the interaction to be more representative of the combined system. This system can then be used in a variety of unknown environments by leveraging data that is easily attainable by simulation. The authors test their approach on 4 different low-dimensional testing environments. They discovered that their approach was comparable to using the true model parameters in each scenario and in some cases performed better.

2.3 System Identification and State Estimation

System identification is the task of accurately modeling the behaviors of a system on the basis of observing input-output relationships [Ljung, 1999]. The general approach is to strategically sample the system, collect the outputs, determine the class of models that represent the relationship well, and modify the parameters of the model to accurately fit the input-output relationship. There are three main types of system identification frameworks that are known as white box modeling, black box modeling, and grey box modeling. These differ in the amount of prior knowledge the modeler injects into the modeled representation of the system. System identification techniques are well studied and various techniques exist depending on the demands of the system approximation [Sjöberg et al., 1995]. Our work is a data driven black box approach using a parametric model to model non-linear relationships.

State estimation seeks to continuously make estimates of the dynamics of a

system through feedback and observing input-output relationships [Simon, 2006]. A common approach to this is using a technique called Kalman filtering. This technique is an iterative process which uses previous state distribution and uncertainty modeled by a covariance matrix with sensor readings with uncertainty of its own. A new state distribution and uncertainty is estimated and the process repeats. Many versions of this filter exist to handle various conditions. State estimation with RNNs has been compared with Extended Kalman Filters (EKF) [N. Yadaiah, 2006]. The authors found that in their scenario the RNN based state estimator was more accurate than the EKF implementation. They develop an architecture which cascades a RNN layer into a Feed Forward Neural Network (FFNN). The motivation is that the RNN layers would represent the system state dynamics while the FFNN transforms the dynamics into measurements. This is similar to our architecture, where the we cascade RNN layers into FFNN layers for final estimation. However, we use the modern LSTM architecture with higher input dimensions.

2.4 Prediction from Time Series

Prediction from time series data has been studied and applied extensively in many fields including economics, biology, linguistics, and industrial settings [Debasish Sena, 2015; Socher et al., 2013; Yazdani, 2009; Ziv Bar-Joseph, 2012]. Using time series data allows for deeper analysis of the behavior of a system. A popular choice for time series prediction is a class of models known as RNNs [Connor et al., 1994; Gers et al., 2000; Prasad and Prasad, 2014]. These models enable accurate and robust data driven approaches to prediction problems which address the issues of highly nonlinear, non-stationary, noisy data. As such, these models are very promising in their capabilities of modeling complex systems. Similar applications to this thesis leverage RNNs for control or state estimation tasks [Alanis et al., 2011; HuH and Todorov, 2009]. By training the RNN to emulate specific behaviors, the authors were able to provide generalized solutions to their task domains. Another use case of RNNs for control is to preserve sequential observation-action sequences to solve Partially Observed Markhov Decision Processes (POMDP) in a scalable way to solve tasks in the context of reinforcement learning [Heess et al., 2015]. By using this model, the authors were able to hide state information such as velocity

from the model in order to allow the model to infer that information to achieve tasks of balancing an inverted pendulum and cartpole swing-ups. Furthermore, time dependent environmental changes could also be introduced and handled gracefully by this model while a non-temporal model such as a feed forward neural network performed poorly.

Chapter 3

Background

This chapter reviews a number of the key components of the simulation and learning system that we develop in this thesis. Figure 3.1 displays a high level overview of the entire system. Each component will be discussed in detail beginning with the walking controller, physics environment, and finally the learning model. At a high level, we apply a physics based bipedal controller to the terrain generated inside our physics simulation to gather data. This data is then given as input to our estimation model and then fed back to the physics simulation for real time estimation of the environment.

3.1 Bipedal Walking Control Policy

Virtual bipedal characters can be animated using physics, leading to rich and realistic interactions with their environment that would be difficult to achieve using traditional animation methods that ignore physics. Accomplishing this requires generating actuation patterns that enable the physics-based character to accomplish its tasks. There are three design approaches that have been explored for generating such patterns. These include Pose-driven feedback control, dynamics-based optimization control, and stimulus-response network control. In a pose-driven approach, target trajectories are proposed for the character which the controller uses to minimize the distance between its current measured state and the target states. In a dynamics-based optimization control approach, actuator torques are proposed

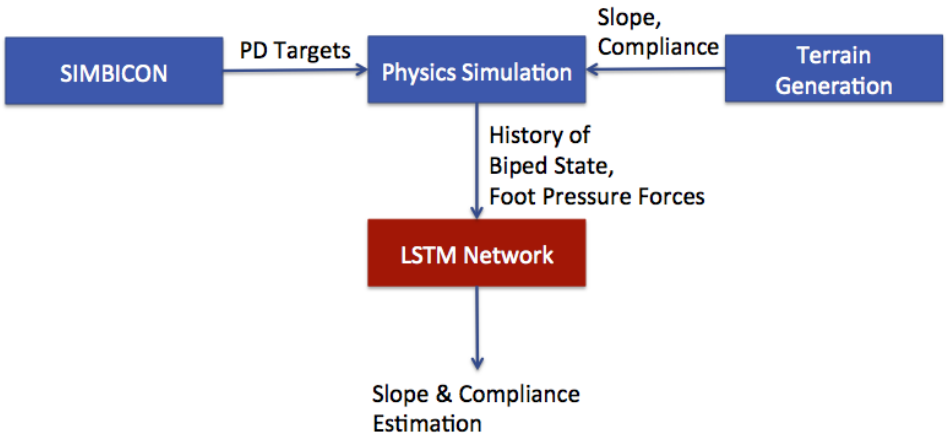


Figure 3.1: System Diagram. SIMBICON Walking controller generates data from physics simulation by being applied on the environment. This data is then used to train a learning model, where the estimations can be displayed in real time.

which try to optimize a set of high level objectives while obeying constraints. Compared to the previous approach, this method establishes a definitive causal effect of torques. However, these methods are computationally more expensive since a system of equations have to be solved at regular intervals, though at a lower rate than the physics time step of the simulation. Finally, a stimulus-response network control seeks to map stimuli from the environment to actuations. These methods do not assume any a priori knowledge and seeks to build the mapping through repeated simulation and evaluation of a fitness function. However, developing and tuning this fitness function to produce natural motion for humans and animals remains elusive and requires extensive knowledge from a variety of different fields.

To that end, we require an effective and robust control approach for bipedal locomotion. This locomotion needs to be robust enough to handle slight terrain perturbations. We settle on the well-known bipedal control policy known as SIMBICON, which is a pose-driven feedback control design [Yin et al., 2007]. This control strategy allow for customization of a wide range of gaits and styles that are robust enough to handle small unexpected forces, terrain changes, and dynamics. It controls characters by defining targets for the joint angles and uses Proportional

Derivative (PD) feedback controllers to compute torques that minimize the difference between the target pose and the current pose.

The controller is based on a Finite State Machine (FSM) along with a feedback mechanism for determining swing hip position. The FSM can be altered to produce different style gaits while remaining robust towards small perturbations. Each state provides a set of target joint positions and velocities for each individual joint. FSM state transitions either when the time duration of the state has been reached or a contact has been made with the surface as shown in Figure 3.2. PD controllers are used to compute joint torques as computed according to $\tau = k_p(\theta_d - \theta) + k_d(\dot{\theta}_d - \dot{\theta})$. Here, θ_d is the desired orientation of the child link with respect to its parent link, θ is the current orientation, $\dot{\theta}_d$ is the desired angular velocity, $\dot{\theta}$ is the current angular velocity, and k_p and k_d are gains for position and velocity respectively. Typically the desired angular velocity $\dot{\theta}_d$ is set to 0. Furthermore, there is a feedback law specific to the swing hip that modifies the target orientation based on the stance foot distance to the Center Of Mass (COM) and its linear velocity in order to keep the character stable. This feedback component is computed according to $\theta_d = \theta_{d0} + c_d d + c_v v$. Where θ_{d0} is the original desired orientation, d is the distance between the hip and the stance foot, v is the linear velocity of the hip, and c_d and c_v are gain parameters. Virtual torque is applied to the torso to keep it upright with respect to the world frame, which is realized through the hips so that it can be produced via internal torques.

For our purposes, we require a basic walking gait that is able to traverse variations in terrain. These variations take the form of ground slope and ground compliance. A walking gait robust to these perturbations is found through manual selection of the parameters as based on the original SIMBICON paper [Yin et al., 2007]. Further details are given in Chapter 4.

3.2 Ground Contact Model

We require a ground contact model that can provide the information needed to model the sensing of pressure at various predetermined points along the sole of the foot. These pressures are assumed to be available as part of the sensory stream. we model ground compliance using dynamically instanced springs and dampers,

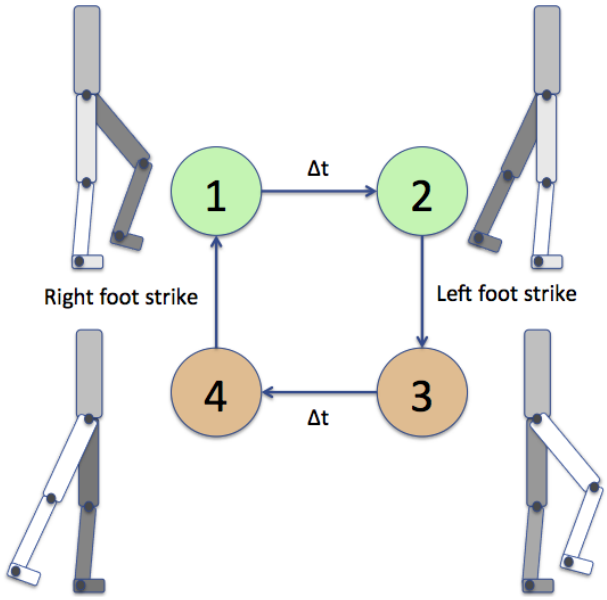


Figure 3.2: Example Finite State Machine for SIMBICON. The state transitions that exit states 1 and 3 occur after a time delay, Δ t. States 2 and 4 are completed up until the corresponding foot has made contact. States 1 and 2, shown in green, are in right stnace, while states 3 and 4, shown in orange, are in left stance.

which apply restorative forces to specific points on a rigid body in order to keep it above the ground.

Figure 3.3 shows an example of the forces being applied to a set of vertices on the foot. Each restorative force is computed using a spring and damper according to

$$F_r = k_p(C_p - P) - k_d\dot{P}$$

where C_p is the point of first contact with the ground, P is the current position of the vertex, $\dot{P}$ is the velocity of the vertex, and k_p, k_d are stiffness and damping constants. A closeup of the spring and damper model is shown in Figure 3.4. In order to avoid creating unrealistic forces that pull the foot down, we further filter the vertical reaction force according to $F_y' = \min(0, F_y)$.

We apply a friction cone to limit the allowable directions that the forces can be directed according to the coefficient of friction as defined by $\mu = \tan\theta = \frac{|F_t|}{|F_n|}$

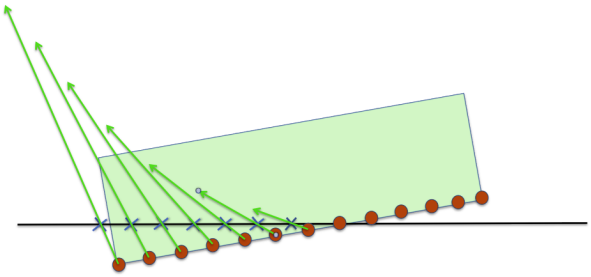


Figure 3.3: The rigid body is drawn as a light green rectangle. Red circles represent vertices of rigid body that will be used for collision detection and response. X's represent the location of first contact with the ground point. Green arrows represent the restorative force vectors applied to the vertices.

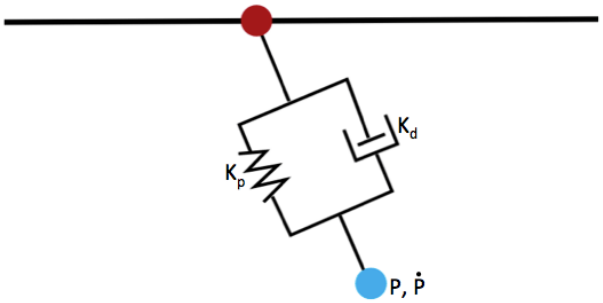


Figure 3.4: Spring and Damper system used for the contact model.

where F_t is the tangential force component, and F_n is the normal force component as shown in Figure 3.5. Once the restorative force is calculated for each vertex, we compare it with the allowable ranges specified by the friction coefficient. If the force exceeds the bounds of the friction cone, we clip the force to the outside edge of the cone and move the collision point to reflect this change, i.e $|F_t| = \min(|F_t|, \mu * |F_n|)$. The forces felt on the feet are shown in the blue lines in Figure 1.2

3.3 State Estimation and System Identification

The problem we seek to solve is related to the task of determining the state of a dynamical system, which can be generalized to also model the state of the envi-

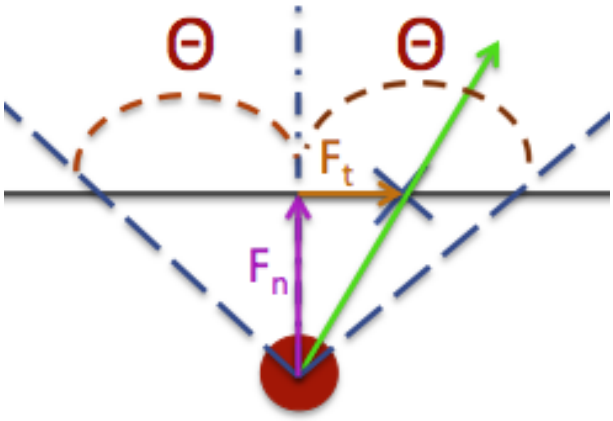


Figure 3.5: The rigid body is represented as a light green rectangle. The circle represents a vertex of rigid body to track. The X represents the collision point of the vertex with the surface. The green arrow represents the restorative force vector applied to the vertex.

ronment. Accurate knowledge of the state of the system is crucial for stabilization through state feedback. In our case, we are interested in using a series of character state measurements observed over time as a means to better approximate the environmental state. Our approach is loosely related to what the Kalman filter seeks to accomplish with continuous states. The Kalman filter recursively predicts and updates its current estimates and covariances with corrections from observed measurements at each step. The predict step is characterized by predicting the next estimate of the state as well as the uncertainty modeled by the updated covariance. Different variations of the Kalman Filter are used ubiquitously in the field of robotics for state estimation depending on the type of system encountered. Examples include applications estimating joint friction of bipedal walkers, vehicle states, and robot positioning [Hashlamon and Erbatur, 2016; N. Houshangi, 2005; Reina et al., 2017].

3.4 Neural Networks

Neural Networks are hierarchical models which learn useful representations of the data at each layer. Each layer consists of a particular model that extracts its own

purposeful and useful representations from its input. Neural Networks are well known as flexible function approximators that can be trained end-to-end through supervised learning. Because of their success in accurate generalization, these models are applied in fields including linguistics, computer vision, graphics, control, and biology.

3.4.1 Feed Forward Networks

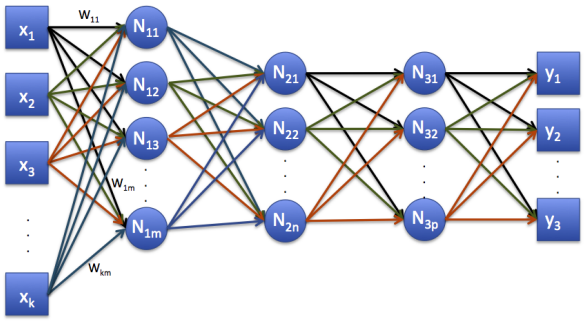


Figure 3.6: Example architecture of a dense feed forward network.

Traditional neural networks are feed-forward neural networks, which have a fixed input size and are fully connected. Figure 3.6 displays an example of a classical Feed Forward Deep Neural Network. Each neuron in the previous layer is connected to a neuron in the subsequent layer. Neurons, represented as circles, compute a weighted sum of its inputs and are typically followed by a subsequent non-linear activation function to produce an output. This is expressed as:

$$a_i^j = \sigma\left(\sum_k w_{ik}^{j-1} a_k^{j-1} + b_i^j\right) \tag{3.1}$$

where a_i^j is the activation value of the i^{th} neuron in the j^{th} layer, σ is the activation function, w_{ik}^{j-1} is the weight that connects the k^{th} neuron to the i^{th} neuron in previous layer $j-1$, and b_i^j is the bias of the i^{th} neuron in layer j. As more layers are added, the complexity of the model grows because the parameter space increases. As model complexity grows, the risk of overfitting also increases.

In order to train these models, we require a loss function which represents

the deviation between the desired values and the values predicted by the network. The backpropagation algorithm is used for training the network by computing the gradients of the loss function with respect to each of the parameters of the network by taking advantage of the chain rule of calculus. Four equations are used for backpropagation:

$$\delta^L = \nabla_a C \odot \sigma' \left(\sum_k w_{ik}^{L-1} a_k^{L-1} + b_i^L \right) \tag{3.2}$$

$$\delta^j = \left((w^{j+1})^\intercal \delta^{j+1} \right) \odot \sigma' \left(\sum_k w_{ik}^{j-1} a_k^{j-1} + b_i^j \right) \tag{3.3}$$

$$\frac{\partial C}{\partial b_i^j} = \delta_i^j \tag{3.4}$$

$$\frac{\partial C}{\partial w_{ik}^j} = a_k^{j-1} \delta_i^l \tag{3.5}$$

Equations 3.2 and 3.3 describe how to compute the gradients of the activation functions with respect to any layer. $\odot$ represents the Hadamard product. In all cases, C is the loss function, w_{ik}^j represents the weights connecting the k^{th} node to the i^{th} node in layer j, a represents the activation value, and b represents the biases. In Equation 3.2 and 3.3, the gradient with respect to the node is stored as δ since all the parameters of the network require this value. Equation 3.2 computes the gradient with respect to the node in the last layer, while equation 3.3 computes the gradient with respect to an arbitrary node in layer j. Equations 3.4 and 3.5 describe how to compute the gradients with respect to each parameter given the layer gradients. Finally each parameter is updated via gradient descent according to

$$\theta' = \theta - \alpha \frac{\partial C}{\partial \theta}$$

where θ represents the parameter to be updated, and α is the learning rate. The update shifts the parameter towards minimizing the cost function.

Having overly complex models allows the possibility of overfitting, which is an issue described as having a minimal training error but poor generalization performance. In order to combat this, we employ a regularization technique known as

dropout. In dropout, random node activations are taken away during training with some probability. This technique can be interpreted as model averaging where the network sees a high volume of different possible network architectures and averages their outputs at test time. Though not necessarily required when copious amount of training data is available, we employ this regularization technique to ensure that the network does not overfit.

3.4.2 Recurrent Neural Networks (RNNs)

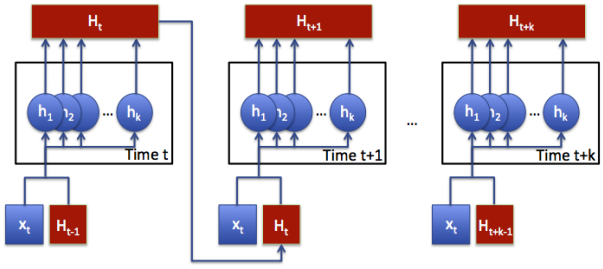


Figure 3.7: Example architecture of an unrolled dense Recurrent Neural Network. Inputs are combined with the previous hidden state and the result is linearly combined before being passed through an activation function. h_i represent the neurons with weighted inputs followed by the activation function.

RNNs differ from traditional feed-forward neural networks in their ability to store and propagate internal memory across arbitrary length sequences of inputs. These types of networks specialize in retaining time dependencies hidden in the input sequence.

3.4.3 Vanilla RNN

Figure 3.7 shows a basic architecture of an unrolled vanilla RNN. These networks operate sequentially on one input at a time with the same parameters and compute a hidden state at each step. This makes learning much faster compared to traditional networks because the number of parameters is much smaller than a feed-forward network that receives the entire temporal sequence of data. The hidden values are then passed forward in time and combined with the next input. The final output of

the network can be viewed as another layer which uses the hidden state as an input to an activation function. Formally, the hidden state value is computed as:

$$h_i = \sigma\left(Ux_t + WH_{t-1} + b_i\right) \tag{3.6}$$

where h_i represents the hidden activation at time t for the i^{th} hidden unit, U is a vector of weights to multiply the input, W is a vector of weights that is multiplied with the previous hidden state, b is the bias for the hidden unit, and σ is an activation function. A common activation function is the tanh function which compresses the output to the range $[-1,1]$. At the beginning of each sequence, the hidden state can be reset to an initial value.

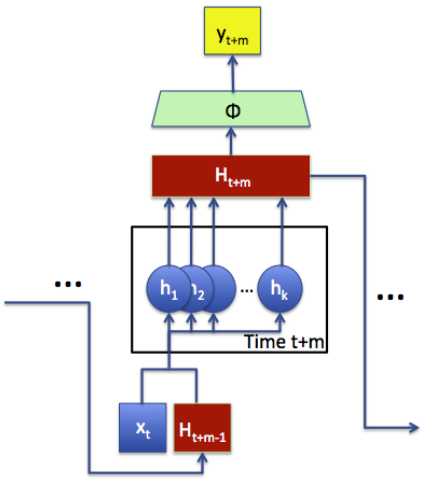
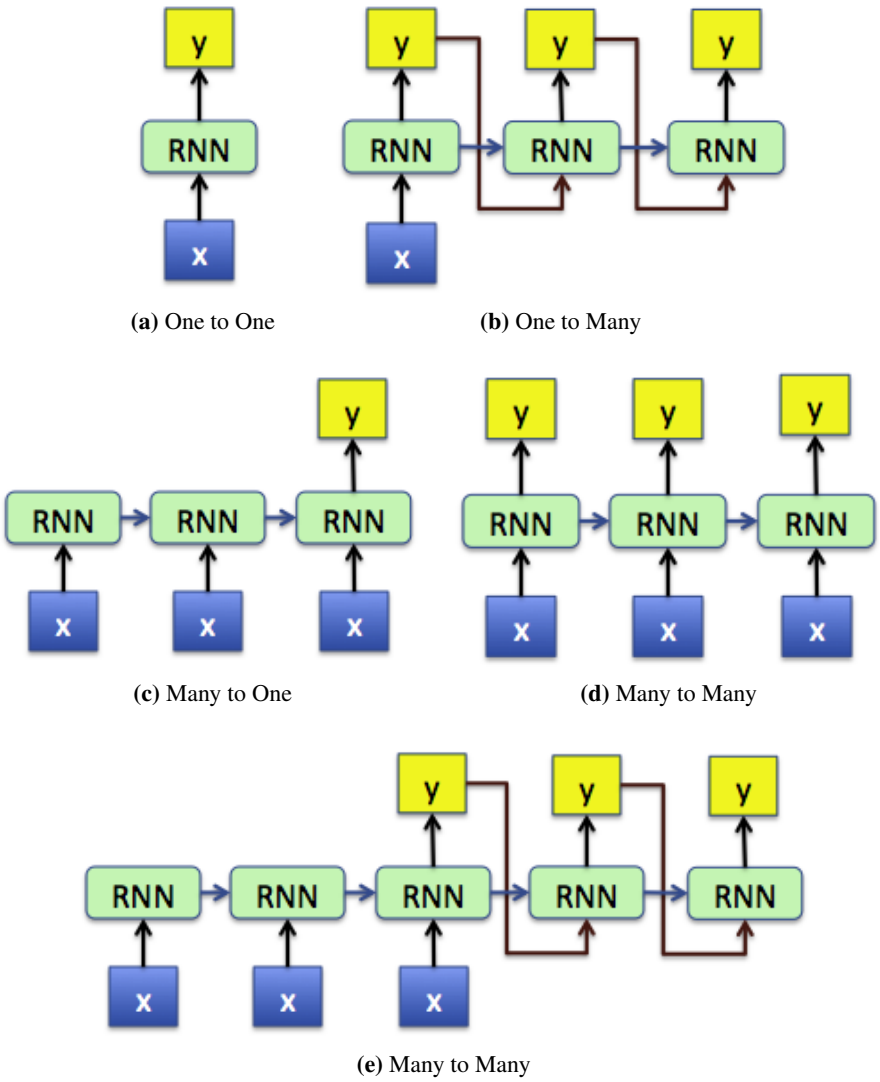


Figure 3.8: Function ϕ transforms the hidden state into an output. This operation can be performed at any stage in the sequence.

The power and flexibility of these networks lies in leveraging the hidden states to produce meaningful outputs. Figure 3.8 shows an example of how to use the hidden state to compute an output. Using this capability, the RNN is able to provide many forms of sequential output. Figure 3.9 shows the various forms that RNN outputs can take. The output form of the RNN depends on the task it is trying to accomplish.

There are several methods for training a RNN. By far, the most popular one



(a) One to One **(b)** One to Many

(c) Many to One **(d)** Many to Many

(e) Many to Many

Figure 3.9: Various RNN Output forms

is the Back Propagation Through Time (BPTT) algorithm, which is what we use to train our network. The BPTT algorithm is essentially the same as the standard backpropagation algorithm on the unrolled RNN network. The difference is that the same parameters of the RNN are used at every step and each gradient contribution at each time step is summed together. As with deep feed-forward networks, gradients

may vanish or explode when propagated through a long sequence since gradient multiplications results in small or large values as more multiplication operations take place depending on the magnitude of the weights and activation function types [Pascanu et al., 2012]. These gradients pose a problem because exploding gradients cause instabilities in the training and vanishing gradients make learning long term dependencies difficult. There are numerous techniques to deal with exploding gradients such as time step truncation and gradient clipping, but vanishing gradients are much more difficult to remedy. In order to solve this problem, a different architecture of the Recurrent Neural Network was conceived, known as the LSTM [Lipton, 2015].

3.4.4 LSTM (Long Short Term Memory) Cells

LSTM networks have the same capabilities of the vanilla RNN. However, LSTM networks solve the vanishing gradient problem by introducing a memory cell state with gates that dictate when the cell state is written to and read from through the use of input and output gates [Hochreiter and Schmidhuber, 1997].

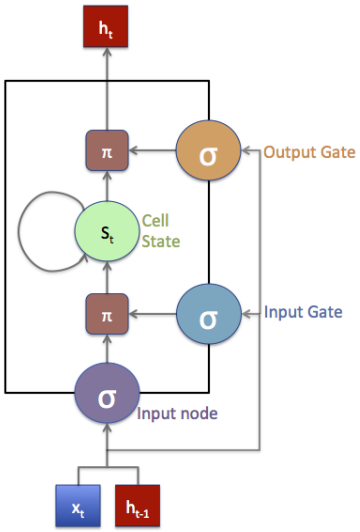


Figure 3.10: LSTM cell proposed by Hochreiter. This architecture introduces the input and output gates which regulate the cell's hidden state.

Figure 3.10 shows the architecture of the LSTM cell proposed by Hochreiter

in 1997 [Hochreiter and Schmidhuber, 1997]. In this figure, the input and previous hidden state are linearly combined and fed into the input node, the input gate, and the output gate. These gates have non-linear activation functions that operate on the inputs. In order to reach the cell state, the input node is multiplied elemental-wise by the input gate. This restricts the information that is allowed to modify the cell state. The cell state keeps its internal state across many iterations by having its own self recurrent connection. The equations dictating the LSTM cell is specified as:

$$I_t = \sigma(Wx_t + Uh_{t-1} + b_i) \tag{3.7}$$

$$N_t = \sigma(Wx_t + Uh_{t-1} + b_n) \tag{3.8}$$

$$C_t = G_t \odot I_t + C_{t-1} \tag{3.9}$$

$$O_t = \sigma(Wx_t + Uh_{t-1} + b_o) \tag{3.10}$$

$$h_t = O_t^g \odot C_t \tag{3.11}$$

where N_t is the input node, I_t is the input gate, C_t is the hidden cell state, O_t is the output gate, and h_t is the hidden state that is propagated forward. Having the self recurrent connection C_t allows for the error to be propagated back for long depths. σ represents a non-linear function and is usually the tanh function. Having a recurrent connection which spans adjacent time steps with a constant weight allows errors to be back propagated without vanishing or exploding. This is known as the Constant Error Carousel (CEC). More specifically, because the cell state is linearly combined, the gradients from the upper levels of computation can flow directly down to the lower levels of the cell. The input and output gates enable the LSTM cell to selectively update or read from its cell state.

Since its first conception in 1997, there has been a recent addition of another gate known as the forget gate [Gers et al., 1999]. The motivation for this addition is to give the LSTM cell the ability to flush its cell state to discard irrelevant information thereby learning to forget. This gate modulates the previous cell state to produce a new cell state. The modern LSTM cell is shown in Figure 3.11.

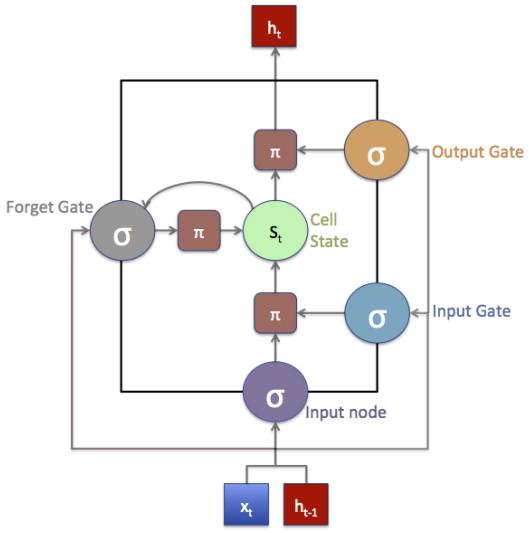


Figure 3.11: Modern LSTM cell with forget gate proposed by [Gers et al., 1999]. The Forget gate allows LSTM cell to flush out irrelevant contents from the cell state while maintaining the constant error carousel.

The forget gate and the new cell state equations are:

$$F_t = \sigma(Wx_t + Uh_{t-1} + b_f) \tag{3.12}$$

$$C_t = G_t \odot I_t + F_t \odot C_{t-1} \tag{3.13}$$

As with the RNN there can be multiple hidden units that make up the hidden state. If the dimension of h_t is k, then the concatenated input is $d+k$ where d is the dimensionality of the state vector. Each gate and node receives its own copy of the inputs which are densely connected to each hidden unit. Thus there are approximately $4k(d+k)$ parameters per LSTM cell. The LSTM network is likewise trained using the BPTT algorithm.

Chapter 4

Methodology

In this chapter we discuss how the physics simulation is set up, the architecture of the LSTM network, as well as the training process of the LSTM network.

4.1 Physics Simulation

As previously mentioned, we use the Bullet physics engine to develop our simulation. We use a time step of 0.0005s in order to achieve stable simulation while maintaining real-time performance.

4.1.1 Biped Simulation

To drive the biped, we compute torques using PD controllers along with a feedback component for calculating swing hip angle. These torques are applied before each

Body	Mass (kg)	Length (m)
Torso	70	0.48
URL	5	0.45
ULL	5	0.45
LRL	4	0.45
LLL	4	0.45
RF	1	0.25
LF	1	0.25

Table 4.1: Body parameters

Body	k_p	k_d
Torso	4200	420
URL	800	80
ULL	800	80
LRL	700	70
LLL	700	70
RF	80	8
LF	80	8

Table 4.2: Joint PD Gains

Parameter	State 1,3	State 2, 4
Δt	0.31s	contact
C_d	10	10
C_v	4	4
Torso (WF)	1.5°	1.5°
Swing hip	46.0°	-7.0°
Swing knee	68°	17°
Swing ankle	-9°	-5°
Stance knee	8°	15.7°
Stance ankle	-7°	-3.0°

Table 4.3: Finite State Machine parameters, WF represents with respect to World Frame

physics step. The body parameters of the 2D Biped are shown in Table 4.1, while joint parameters are shown in Table 4.2. The finite state machine parameters are shown in table 4.3. The segments are connected using revolute joints with joint limits. There is also a maximum allowable torque of $300Nm$ per application.

As previously noted, the ground compliance model that we implement is based on a spring-and-damper restorative forces. The compliance model consists of a stiffness gain, k_p and a damping gain, k_d. We set $k_d = 0.1k_p$. The stiffness is sampled uniformly from a range $k_p \in [1000, 3000]Nm$. We apply this spring system to 22 vertices located on the bottom edge of each foot of the biped. We found that using more vertices along the feet offered more robustness to the gait. Figure 1.2 shows the force vectors in blue, that act upon the vertices along the bottom of the foot.

4.1.2 Terrain Generation

The terrain that the biped traverses follows repeated instances of 3-segment chunks. Each segment is 4 meters in length. The first segment uses a slope $m_1 \in [-5^\circ, 5^\circ]$. The second segment receives a slope $m_2 = 0^\circ$. The last segment's slope is given by $m_3 = -m_1$. The purpose of the flat terrain segments, i.e., $m_2 = 0$, is to give the biped the opportunity to reset its gait. Figure 4.1 displays an example of the terrain pattern that the biped is traversing.

Figure 4.1: The slope pattern that the bipedal character traverse across.

4.2 Network Architecture

For our experiments, we employ a 2 layer LSTM network with a dense feed forward network with a Rectified Linear Unit (RELU) before the final output. Selecting this architecture was performed while varying hyper-parameters and verifying the validation loss. Deeper LSTM layers are helpful for more complex relationships. More parameters allow for modeling of more complex relationships but leaves the possibility of overfitting. Finding an architecture which generalizes well depends on the complexity of the problem and the number of parameters should scale accordingly. Having a Dense Feed-Forward network at the end of the model transforms the hidden state into the estimation. For our intuition, the first LSTM layer transforms the input sequence into a hidden sequence that the model learns to understand well. The second LSTM layer takes this hidden sequence and learns to recognize important data representations which is then decoded by the final dense layers. Figure 4.2 displays the architecture that we use in our experiments.

Our framework uses the many-to-many output form of the RNN as shown in Figure 3.9d for training to take advantage of all the corresponding labels for each time step in the sequence. When testing, we use the many-to-one output form as shown in Figure 3.9c which predicts the terrain properties after the last input from the sequence has been consumed. The outputs of the model are linear units because

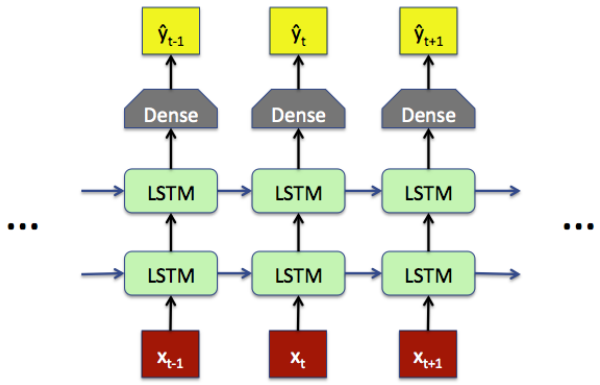


Figure 4.2: The standard multilayer LSTM architecture

our problem resembles a regression.

4.3 Training Process

The training process consists of data collection and offline training of the model. Data collection is the process of consolidating the simulation results which include the biped character state as well as the terrain labels. Our training process consists of sampling batches of data, providing the corresponding labels, and updating the model.

4.3.1 Data Collection

We collect consecutive input vectors along with their associated labels. The input vector includes features representing the torso's linear velocity, each bodies distance to the root (hip joint), orientation, angular velocity, as well as an averaged window of forces. In total, our input vector has 45 dimensions. We gather the labels for the terrain based on the position of hip joint with respect to the ground. That is, the label for the state is determined by which ground segment the hip position is currently over. Thus, there are cases where the swing leg is ahead of the hip and contacts the next terrain segment first while the hip is still over the current terrain segment. These situations give the model a notion of anticipation where it observes a change in terrain prior to its complete arrival. Along with our proposed

network model and state input, we experiment with different window lengths and state features as well. In total, our state vector has a maximum of 45 dimensions.

We found that sampling at greater frequencies allowed for more accurate results. For our simulation we sample at approximately 30Hz. Each sample consists of a state vector which includes the biped state as well as the foot pressure forces computed by the compliance model. Each state vector includes the linear velocity of the hip, each body's orientation and angular velocity, and the restorative force vectors acting upon each vertex along the bottom edge of the foot. We run the simulation on an Intel Core i5 CPU at 2.66 GHz.

4.3.2 Training

Once we have collected the data, we begin our offline training of the model. During preprocessing, we normalize the inputs and the outputs to have zero mean and unit variance. We then employ an overlapping windowing strategy to present sequences to the network. Each window includes 30 consecutive states and an average foot pressure forces. This corresponds to roughly having a 1 second stream of data that is given to the model for estimation. Because the data does not fit into RAM, we select batches of sequences uniformly during training. Our network uses the mean squared error loss along with the RMSProp optimizer [Tieleman and Hinton, 2012]. Furthermore, it should be noted that for final test prediction, we normalize test values with respect to the training values using the mean and and variance from the training data.

Chapter 5

Experiments

This chapter describes the experiments that we perform. Each experiment tests different architectures and input state configurations. In the following sections, we describe our training framework and the evaluation results we use to gauge performance.

5.1 Parameter Settings

For all of our experiments, we use a training, validation, and test set framework; the model updates itself on the training set while measuring its performance on the validation set after each epoch. We stop training if the validation error does not improve after a patience parameter (The number of consecutive epochs where the validation loss does not decrease) of 10 or if the maximum number of epochs, 40, is reached. The training is performed using batches of 32 samples for 530 times as sufficient performance was reached using those metrics. Validation losses were computed and aggregated and averaged over every sample in a sequence since we predict on each sample. We then use the model with the smallest validation loss for testing. The final test score is realized by predicting on the test set. Experiments are each run 3 times in order to gain an understanding of the distribution of the final models. Each experiment contains a validation loss graph, a sample test error graph, and a histogram of errors. With regard to the validation loss graph, the arrows represent the minimum validation losses for each trial. With regard to the

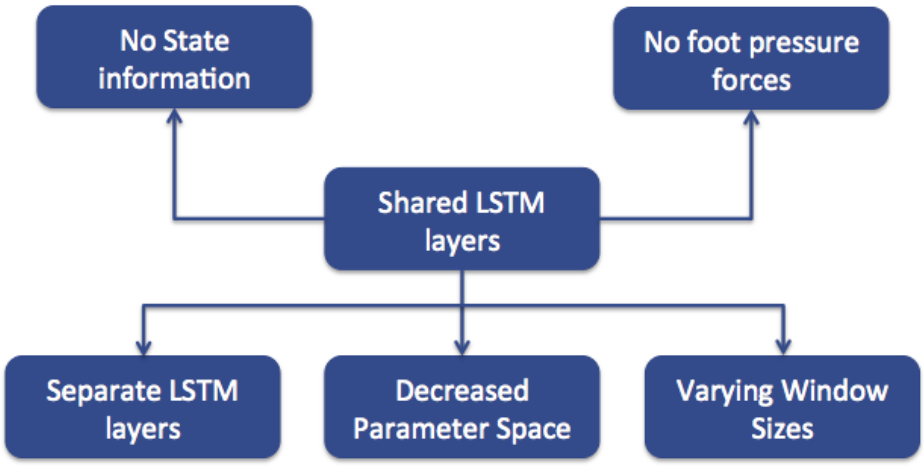


Figure 5.1: The experiments that were carried out. Each experiment branches off from the shared network architecture to measure its effect on accuracy.

sample test error graphs, the blue circles represent the model estimates while the red circles represent the actual values, the black lines show the distance between the actual values and the estimation.-

5.2 Experiments

We begin with the shared network model where the outputs share the same LSTM layers before splitting into separate outputs. Each experiment follows from the base case of using shared LSTM layers to measure its effect on estimation accuracy. We try separate experiments where each experiment consists of a separate model that estimates individual terrain parameters. By doing this, we determine if each estimation benefits from having dedicated parameters. Next, we determine if having fewer hidden units affects the estimation accuracy. Finally, we gauge the effect of different input state configurations. Specifically, we measure the effect of including only bipedal character state and the effect of including only foot pressure forces to estimate terrain properties. Figure 5.1 graphically displays the conducted experiments. To get an idea of the scale of the errors, we note that $0.1\sigma_s = 0.288^\circ$, and $0.1\sigma_c = 57.192Nm$ where σ_s and σ_c stand for slope and compliance standard

deviation respectively.

5.2.1 Shared Network

We introduce the shared LSTM network with separate time distributed fully connected layers as separate outputs for slope and ground compliance shown in Figure 5.2. Each LSTM layer consists of 128 hidden units. Each fully connected layer consists of 128 units. There are 2 linear outputs from the model for slope and compliance. The total number of trainable parameters is 253,954. We use a fixed window size of 30 and predict at every step in the sequence. Figure 5.3a displays the validation losses per trial. For comparison, the validation error of using the average label value for both slope and compliance estimation is 1.892. This value gives an idea of the improvement that this model has over just using the average label. The lowest achieved validation loss is 0.0481.Using the trained model with the lowest validation loss, we run the model on a test set of 2000 samples to get an idea of the estimation accuracy as shown in Figure 5.3b. Here, the blue circles represent the prediction made by the model, the red circles are the actual labels, and the black lines represent the distance of the prediction to the actual labels. We see that the predictions estimate the labels quite closely. There are leading and trailing edges during estimation because the correct label corresponds to the terrain segment that the hip is currently over, which have discrepancies since the swing foot is ahead of of the hip and anticipates terrain changes. Finally, to get an understanding of the distribution of errors, we plot a histogram of errors as shown in Figure 5.3c. In both histogram plots, the errors are closely centered around 0, which suggests high fidelity to the actual labels. Note that the test losses for slope are smaller than compliance for two main reasons. One, the slope generation has a specified terrain pattern while compliance generation does not, which means that a terrain slope of 0 occur more frequently than other slopes, giving the model the opportunity to learn its representation and minimize errors. And two, the range of compliance values is much greater than that of slope, which may require more sampling in order for the model to build an accurate representation of the data. Furthermore, to explain the multimodal distribution of the slope, we see that there are much larger errors along the edges of the estimations than between the edges, which gives rise to the small

sharp peaks in the error histogram.

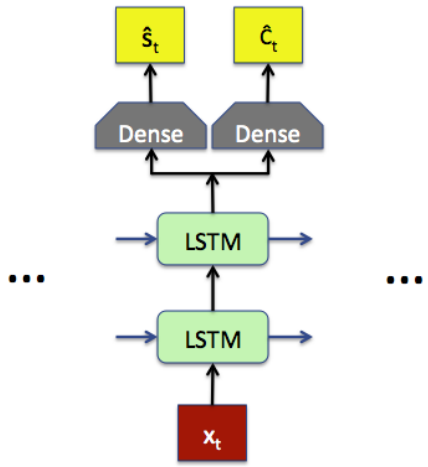


Figure 5.2: The Shared LSTM architecture for predicting both slope $\hat{s}$ and compliance $\hat{c}$.

5.3 Separate Networks

The model that we use for separate slope and compliance predictions is shown in Figure 4.2. Each model is dedicated to estimating one of the terrain properties. Each LSTM layer has 128 hidden units. The Dense layer consists of 128 units as well. The total number of trainable parameters is 237,313. We begin with the dedicated compliance model followed by a dedicated slope model. Table 5.1 displays the validation losses across all trials. The validation loss for the separate models are added to compare with the validation losses from the model with shared parameters. It is interesting to note that the lowest validation loss for the shared model is lower than the combined lowest validation losses of the separate models for slope and compliance.

5.3.1 Compliance

Figure 5.4 displays the results of using a separate network for compliance estimation. Each validation trial where the network is trained from random different parameter initializations. The validation loss for comparison using the average

compliance value as an estimate is 3.484551. The lowest achievable validation loss is 0.031. We then use the model with the lowest validation loss as our final test prediction model. Each test sample represents a sequence of inputs. We take the last predicted output as our final prediction to compare with the ground truth. Note the anticipation towards the next compliance values shown by the large compliance errors towards the edges; the model tries to predict the next values but the hip is still over the current terrain segment. Once the foot pressures are felt for the next segment, the model anticipates the change in terrain and adjusts its estimate. The histogram distribution of errors shows the concentration of errors which is centered around 0, implying accurate prediction. Comparing the test histogram of a dedicated compliance model with the shared model, we see that the test errors of the dedicated compliance model are actually less accurate than the shared model. The dedicated model has a larger mean and standard deviation, implying that there may be information gained from using slope estimations as well.

5.3.2 Slope

We perform the same experiments with slope as we did with compliance. Figure 5.5a displays the validation losses across 3 different parameter initializations. The validation loss for comparison using the average slope value is 0.298591. The lowest validation loss is 0.015. Once we have established the model with the minimum validation loss, we use that model on our test data set. The test estimations are shown in Figure 5.5b. As mentioned previously, since the model sees more examples of 0 slope, it learns that representation very well, keeping the variance of the errors minimal during those stretches. To get a better understanding of the distribution of the errors we plot the error histogram as well shown in Figure 5.5c.

Test	Trial 1	Trial 2	Trial 3
Slope	0.0169	0.0177	**0.0159**
Compliance	**0.0314**	0.0329	0.0320
Shared	**0.0453**	0.0482	0.0509

Table 5.1: Validation losses between the separate and shared models. Lowest validation losses per trial are indicated by their respective arrows

Comparing this histogram of errors with the histogram of errors from the shared model, we see that the distributions look about the same with minor improvements in mean and spread. Thus, we conclude that there is not too much of an improvement using a dedicated model for slope estimation than using a shared model.

5.4 Decreased Parameter Space

We now perform our same methodology using half the number of units in the shared architecture. Each LSTM layer consists of 64 units. Each fully connected layer consists of 64 units as well. The total number of trainable parameters becomes 69,634.

Figure 5.6a displays the validation losses across 3 separate trials from different parameter initializations. Comparing the validation losses with the shared model of more parameters, we see that the shared model achieves lower validation loss consistently across all trials. The lowest validation loss from using half of the units is 0.053. Test predictions are shown in Figure 5.6b. Comparing sample test predictions, we see that the predictions made by this model have slightly more variance. We verify this by comparing the histogram plots of test errors generated by this model and the previous model. The test histogram is shown in Figure 5.7c. Comparing slope error distributions, this model has a mean farther from 0 as well as a larger deviation. When comparing compliance errors, we see that the mean is slightly farther from 0 and the spread is slightly larger for this model. We conclude that the size of the parameter space affects the estimation quality of the model, but more experiments are needed to gauge the relationship between number of parameters, parameter selection, and estimation quality.

5.5 Varying Window Sizes

In this experiment we vary the window sizes of the input data to gauge its effect on the prediction quality of the shared architecture. Our initial experiment with the shared architecture were run with a window of size 30. In this section, we display the results of changing the window to sizes of 15, 7, and 3. We run each experiment with 20 epochs instead of 40 because the validation improvements after 20 are minimal.

Experimental results using a window of 15 are displayed in Figure 5.7. Experimental results using a window of 7 are displayed in Figure 5.8. Experimental results using a window of 3 are shown in Figure 5.9. Comparing these results together, we observe a general trend of loss of predictive quality as window size decreases. The validation losses are larger as window size decreases. From the sample test examples, we see that the variance of the estimates increases as window sizes decreases. This is reflected through the spread of test error which also increases for both the slope and compliance estimations as the window size decreases. This leads us to conclude that using a larger window size indeed provides greater accuracy for estimations. There is relevant information contained in the sequence history that requires a larger window to track it.

5.6 Removal of Features

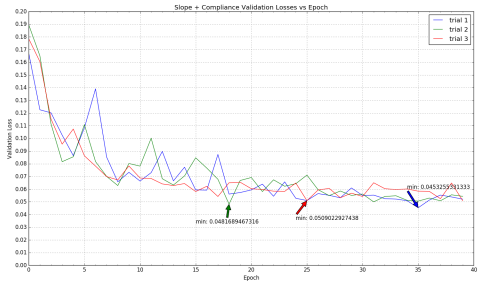
The following experiments gauge the effect that the inputs have on estimation accuracy. In all cases, the model using the entire feature set had higher estimation accuracy and precision. We begin with removing the foot pressure forces from the inputs, thereby only using bipedal character state features to estimate terrain properties. Afterwards, we remove the bipedal character state features and only use foot pressure forces to estimate terrain properties. We then compare the results of these experiments together to draw a conclusion on how each feature set affects estimation.

Results of removing the foot pressure forces and only using state features are shown in Figure 5.10. Results of removing the state features and only using foot pressure forces are shown in Figure 5.11. Comparing the validation losses together, we see that in all cases the validation loss after one epoch with just using the foot pressure forces improved more sharply than that of just using state information. This may be due to the homogeneity of the foot pressure force data. Regardless, the validation losses at the end of training appear similar. By comparing test predictions together, we begin to see some interesting behavior. Using only bipedal character state features, we see that the test predictions of slope fluctuate less and are more accurate than that of just using foot pressure forces for the same estimation. Conversely, when looking at the test predictions for compliance using only

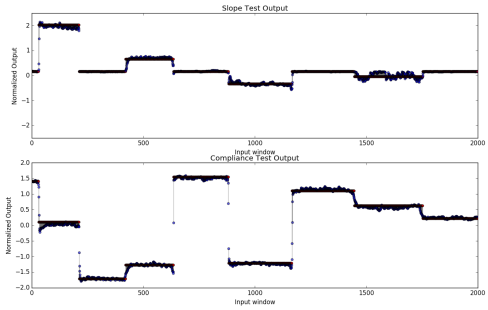
foot pressure forces we see that the estimations oscillate less than that of just using state information for the same estimations. This hints at the notion that bipedal character state captures slope more accurately than using foot pressure forces, and foot pressure forces capture compliance more accurately than bipedal state features. We verify this by comparing the test histograms. By comparing the error histograms for slope from the respective figures, we see that using state results in lower spread for slope with higher accuracy. By comparing the error histograms for compliance, we see that the spread of errors from the compliance estimation using bipedal state features is slightly larger than that of using foot pressure forces. This hints that the model learns to be sensitive to certain features that may capture the desired variable to estimate more accurately.

5.7 Discussion

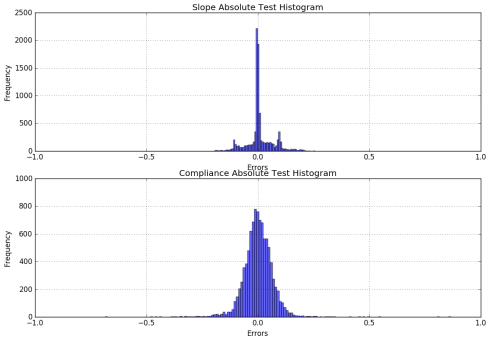
This section provides a brief discussion about the results. We show that a shared architecture gives a sufficient model for our estimation purposes. This model scales the number of parameters with the complexity of the estimation task appropriately. In general, larger window sizes provide better estimates. From our experiments, we believe that slope information is contained in the bipedal state of the character while foot pressure forces contain compliance values. Having memory of the time series inputs that is represented by the cell state of the LSTM cell allows the model to pick up on important patterns hidden in the input sequence. We suspect that by using the sequence of bipedal states, the model is able to pick up on the difference in state values and map those differences to slopes. To make this more obvious, we conducted an experiment (not shown) where we provided a sequence of state differences as time series input to make the information more explicit and verified similar performance quality.



(a) Validation losses for Slope and Compliance predictions. The arrows point to the respective minimum losses of each trial

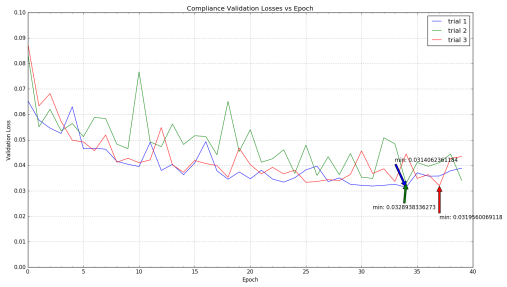


(b) Sample Test Predictions for each output.

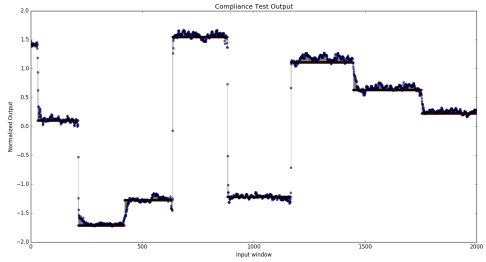


(c) Histograms of test error predictions for slope and compliance. $\mu_s = 0.006, \sigma_s = 0.084, \mu_c = -0.001, \sigma_c = 0.113$

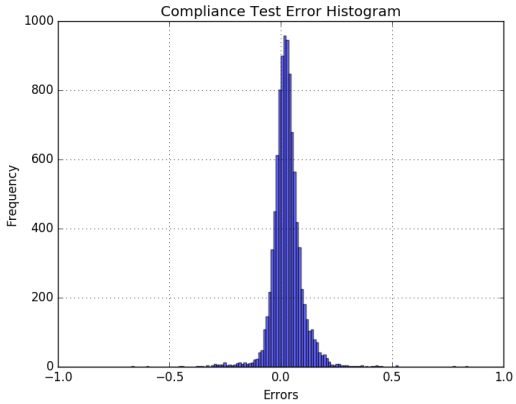
Figure 5.3: Results for Slope and Compliance prediction using a shared architecture



(a) Validation losses for compliance predictions.

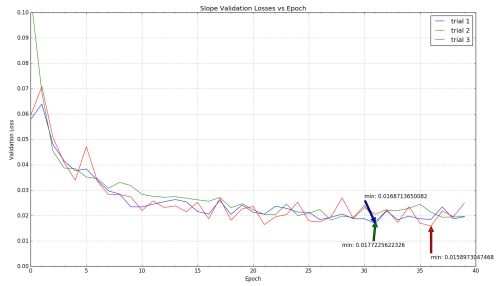


(b) Sample compliance test predictions for each output.

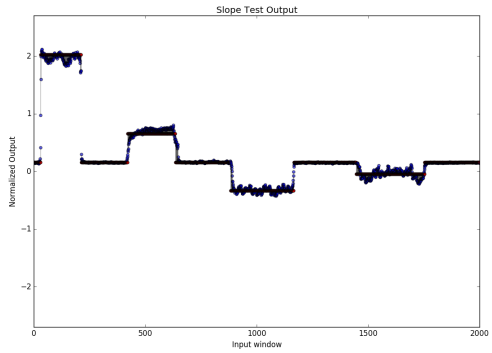


(c) Histograms of test error predictions for compliance.
$\mu = 0.027, \sigma = 0.118$

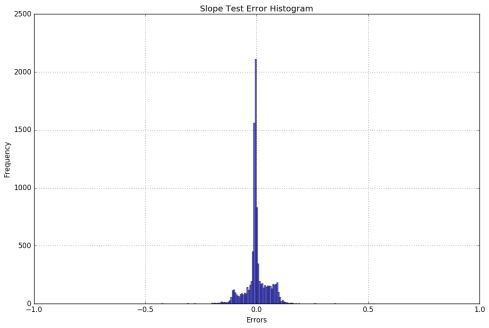
Figure 5.4: Results for compliance prediction using separate model



(a) Validation losses for Slope predictions.

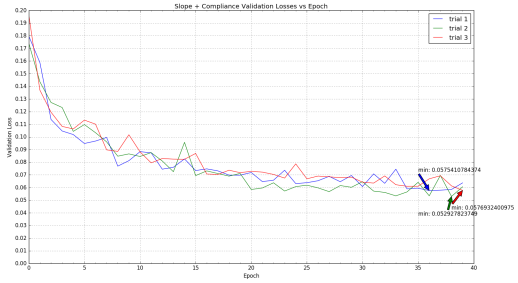


(b) Sample slope test predictions for each output.

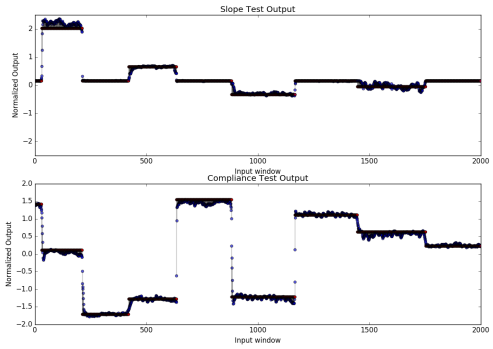


(c) Histograms of test error predictions for slope. $\mu = -0.002, \sigma = 0.072$

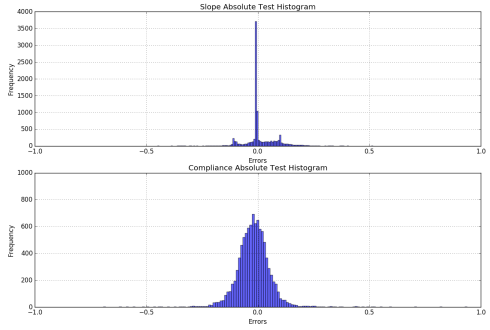
Figure 5.5: Results for slope prediction using separate model



(a) Validation losses for Slope and Compliance predictions.

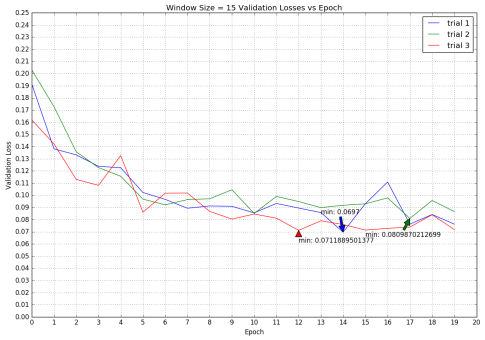


(b) Sample Test Predictions for each output with half of the units.

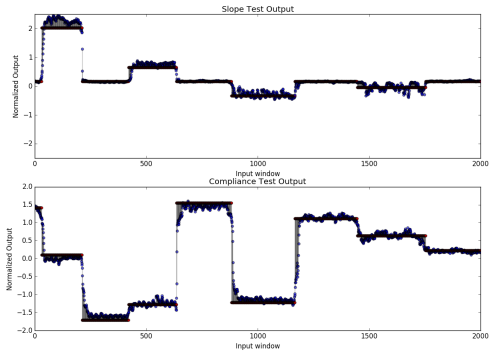


(c) Histograms of test error predictions for slope and compliance. $\mu_s = 0.008, \sigma_s = 0.094, \mu_c = -0.020, \sigma_c = 0.124$

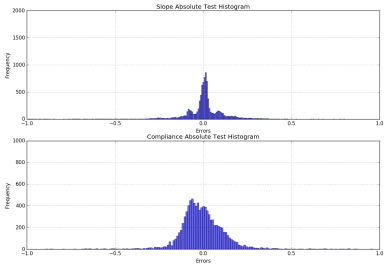
Figure 5.6: Results for using half of the units.



(a) Validation losses for Slope and Compliance predictions for a window of size 15.

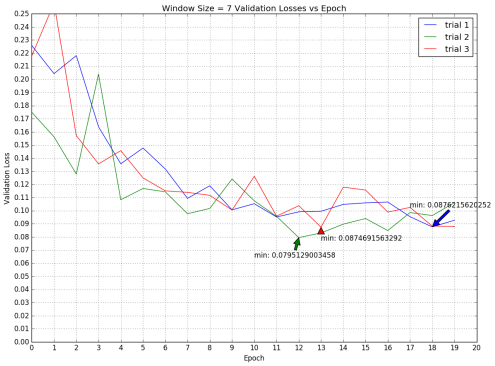


(b) Sample Test Predictions for each output with a window size of 15.

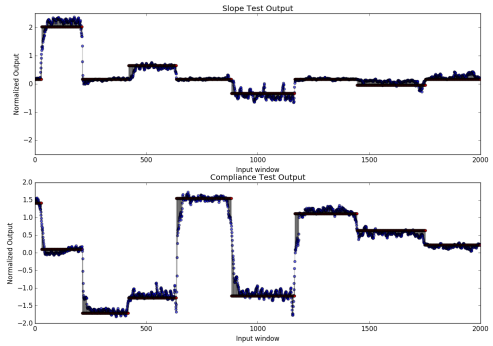


(c) Histograms of test error predictions for slope and compliance for a window of size 15. $\mu_s = 0.001, \sigma_s = 0.144, \mu_c = -0.009, \sigma_c = 0.235$

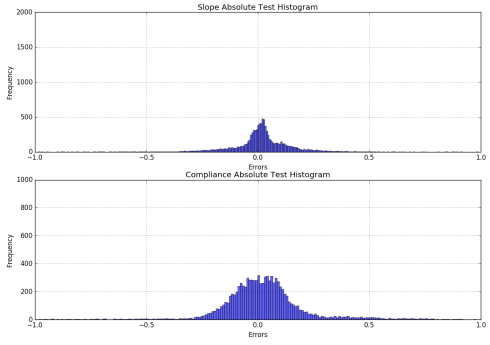
Figure 5.7: Results for a window size of 15



(a) Validation losses for Slope and Compliance predictions.

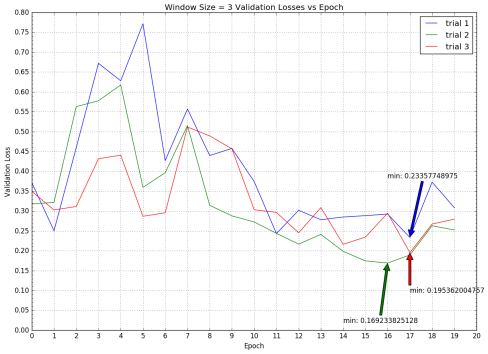


(b) Sample Test Predictions for each output.

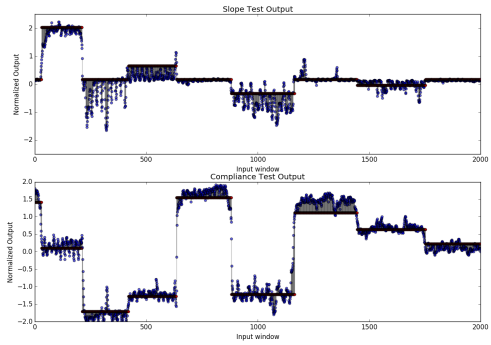


(c) Histograms of test error predictions for slope and compliance. $\mu_s = 0.016, \sigma_s = 0.151, \mu_c = 0.025, \sigma_c = 0.242$

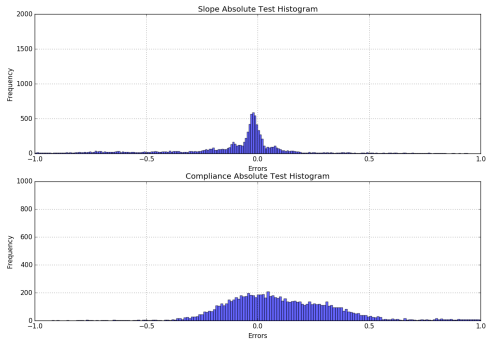
Figure 5.8: Results for a window of 7



(a) Validation losses for Slope and Compliance predictions.

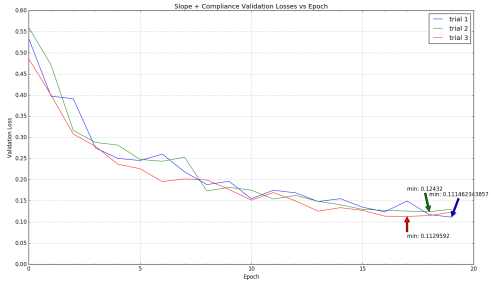


(b) Sample Test Predictions for each output.

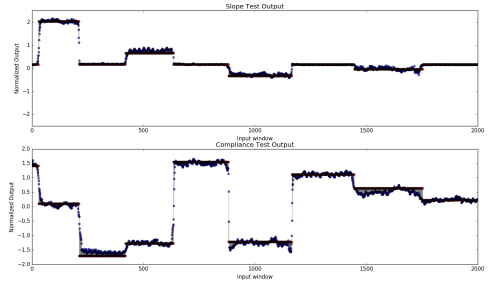


(c) Histograms of test error predictions for slope and compliance. $\mu_s = -0.131, \sigma_s = 0.334, \mu_c = 0.161, \sigma_c = 0.406$

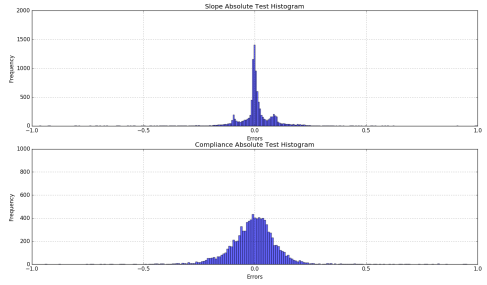
Figure 5.9: Results for a window of 3



(a) Validation losses for Slope and Compliance predictions.

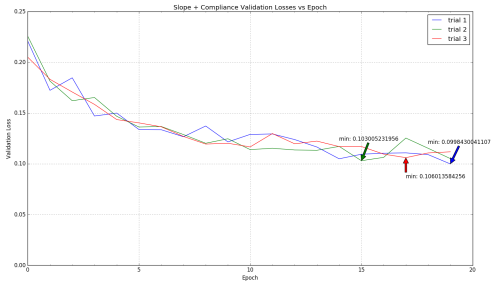


(b) Sample Test Predictions for each output.

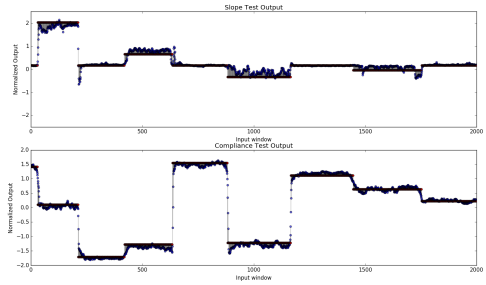


(c) Histograms of test error predictions for slope and compliance. $\mu_s = 0.00178, \sigma_s = 0.120, \mu_c = -0.005, \sigma_c = 0.152$

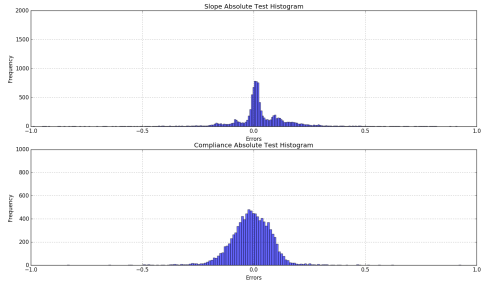
Figure 5.10: Results of only using state features



(a) Validation losses for Slope and Compliance predictions.



(b) Sample Test Predictions for each output.



(c) Histograms of test error predictions for slope and compliance. $\mu_s = 0.028, \sigma_s = 0.146, \mu_c = -0.014, \sigma_c = 0.140$

Figure 5.11: Results of only using foot pressure forces

Chapter 6

Conclusions

In this thesis, we explored using a deep recurrent network along with a recent history of locomotion and foot pressure data for terrain estimation. We were able to develop an architecture with enough complexity that can handle temporal data. Our approach was motivated by the idea of building perception through structured movement and physical sensations gained through the agent's modalities. Through our experiments, we discovered that history length, parameter size, as well as feature type all contribute to the fidelity of the model. Using a longer history allowed the model more opportunity to capture key patterns. Increasing parameter space gave the model more freedom to search for optimal parameter values. Lastly, our experiments hinted that key pieces of information were embedded in the type of features given to the model; the model was able to learn which set of features gave greater correspondences with terrain parameters without any explicit information. Automatically learning these correspondences is similar to how humans learn to pay attention to certain modalities to learn more about certain aspects of the environment. We show that this model achieves high fidelity that can be used in real time as supplemental information. A limitation of this model is the tight coupling of the model with the locomotion, sensory information, and environment; a change in either of these components would require a separate model to be trained. Generalizing a model like this for a variety of motion, terrain, and sensory information is a challenge. In the future, we would like to explore the best use cases for a similar model for boosting controllers. Incorporating a memory element into a controller

may enable it to memorize key events and anticipate important changes allowing for highly dynamic responses by the agent.

Bibliography

Alanis, A. Y., Sanchez, E. N., Loukianov, A. G., and Perez, M. A. (2011). Real-time recurrent neural state estimation. *IEEE Transactions on Neural Networks*, 22(3):497–505. → pages 12

Barsalou, L. W. (2007). Grounded Cognition. *Annual Review of Psychology*, 59:1–21. → pages 6

Berthoz, A. (2002). *The Brain's Sense of Movement*. Harvard University Press. → pages 2

Connor, J. T., Martin, R. D., and Atlas, L. E. (1994). Recurrent Neural Networks and Robust Time Series Prediction. *IEEE Transactions on Neural Networks*, 5(2):240–254. → pages 12

Debasish Sena, N. K. N. (2015). Application of time series based prediction model to forecast per capita disposable income. *IEEE*. → pages 12

Gers, F., Eck, D., and Schmidhuber, J. (2000). Applying lstm to time series predictable through time-window approaches. Technical report. → pages 12

Gers, F. A., Schmidhuber, J., and Cummins, F. (1999). Learning to forget: Continual prediction with lstm. *Neural Computation*, 12:2451–2471. → pages x, 26, 27

Giguere, P. and Dudek, G. (2009). Surface identification using simple contact dynamics for mobile robots. In *2009 IEEE International Conference on Robotics and Automation*, pages 3301–3306. → pages ix, 7

Guo, K., Zou, D., and Chen, X. (2015). 3d mesh labeling via deep convolutional neural networks. *ACM Trans. Graph.*, 35(1):3:1–3:12. → pages 3

Hashlamon, I. and Erbatur, K. (2016). Joint friction estimation for walking bipeds. *Robotica*, 34(7):16101629. → pages 19

Heess, N., Hunt, J. J., Lillicrap, T. P., and Silver, D. (2015). Memory-based control with recurrent neural networks. *CoRR*, abs/1512.04455. → pages 12

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, 9(8):1735–1780. → pages 25, 26

Holden, D., Saito, J., and Komura, T. (2016). A deep learning framework for character motion synthesis and editing. *ACM Trans. Graph.*, 35(4):138:1–138:11. → pages 3

HuH, D. and Todorov, E. (2009). Real-time motor control using recurrent neural networks. In *2009 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, pages 42–49. → pages 12

Lipton, Z. C. (2015). A critical review of recurrent neural networks for sequence learning. *CoRR*, abs/1506.00019. → pages 25

Ljung, L., editor (1999). *System Identification (2Nd Ed.): Theory for the User*. Prentice Hall PTR, Upper Saddle River, NJ, USA. → pages 11

Marion, P., Fallon, M., Deits, R., Whelan, T., Antone, M., McDonald, J., and Tedrake, R. (2015). *Continuous Humanoid Locomotion over Uneven Terrain using Stereo Fusion*, pages 881–888. IEEE. → pages 10

N. Houshangi, F. A. (2005). Accurate mobile robot position determination using unscented kalman filter. *IEEE*. → pages 19

N. Yadaiah, G. S. (2006). Neural network based state estimation of dynamical systems. *IEEE*. → pages 12

Pascanu, R., Mikolov, T., and Bengio, Y. (2012). Understanding the exploding gradient problem. *CoRR*, abs/1211.5063. → pages 25

Peng, X. B., Berseth, G., and van de Panne, M. (2016). Terrain-adaptive locomotion skills using deep reinforcement learning. *ACM Trans. Graph.*, 35(4):81:1–81:12. → pages 3

Prasad, S. C. and Prasad, P. (2014). Deep recurrent neural networks for time series prediction. *CoRR*, abs/1407.5949. → pages 12

Reina, G., Paiano, M., and Blanco-Claraco, J.-L. (2017). Vehicle parameter estimation using a model-based estimator. *Mechanical Systems and Signal Processing*, 87:227 – 241. → pages 19

Sandeep Manjanna, Gregory Dudek, P. G. (2013). Using gait change for terrain sensing by robots. *2013 10th International Conference on Computer and Robot Vision (CRV 2013)*, 00:16–22. → pages ix, 9

Simon, D. (2006). *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*. Wiley-Interscience. → pages 12

Sjöberg, J., Zhang, Q., Ljung, L., Benveniste, A., Delyon, B., Glorennec, P.-Y., Hjalmarsson, H., and Juditsky, A. (1995). Nonlinear black-box modeling in system identification: A unified overview. *Automatica*, 31(12):1691–1724. → pages 11

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A. Y., and Potts, C. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Stroudsburg, PA. Association for Computational Linguistics. → pages 3, 12

Tieleman, T. and Hinton, G. (2012). Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning. → pages 32

Vernon, D. (2008). Cognitive vision: The case for embodied perception. *Image and Vision Computing*, 26(1):127 – 140. Cognitive Vision-Special Issue. → pages 6

Walas, K. (2015). Terrain classification and negotiation with a walking robot. *J. Intell. Robotics Syst.*, 78(3-4):401–423. → pages ix, 8

Wenhao Yu, C. Karen Liu, G. T. (2017). Preparing for the unknown: Learning a universal policy with online system identification. *Pre print*. → pages ix, 10, 11

Wilson, R. A. and Foglia, L. (2017). Embodied cognition. In Zalta, E. N., editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, spring 2017 edition. → pages 2

Yazdani, H. (2009). Prediction of chaotic time series using neural network. In *Proceedings of the 10th WSEAS International Conference on Neural Networks*, NN'09, pages 47–54, Stevens Point, Wisconsin, USA. World Scientific and Engineering Academy and Society (WSEAS). → pages 12

Yi, S.-J., Zhang, B.-T., and Lee, D. D. (2010). Online learning of uneven terrain for humanoid bipedal walking. In *Proceedings of the Twenty-Fourth AAAI*

Conference on Artificial Intelligence, AAAI'10, pages 1639–1644. AAAI Press. → pages 10

Yin, K., Loken, K., and van de Panne, M. (2007). Simbicon: Simple biped locomotion control. *ACM Trans. Graph.*, 26(3). → pages 4, 15, 16

Ziv Bar-Joseph, Anthony Gitter, . I. S. (2012). Studying and modelling dynamic biological processes using time-series gene expression data. *Nature*. → pages 12