Mode-Adaptive Neural Networks for Quadruped Motion Control

HE ZHANG†, University of Edinburgh
SEBASTIAN STARKE†, University of Edinburgh
TAKU KOMURA, University of Edinburgh
JUN SAITO, Adobe Research

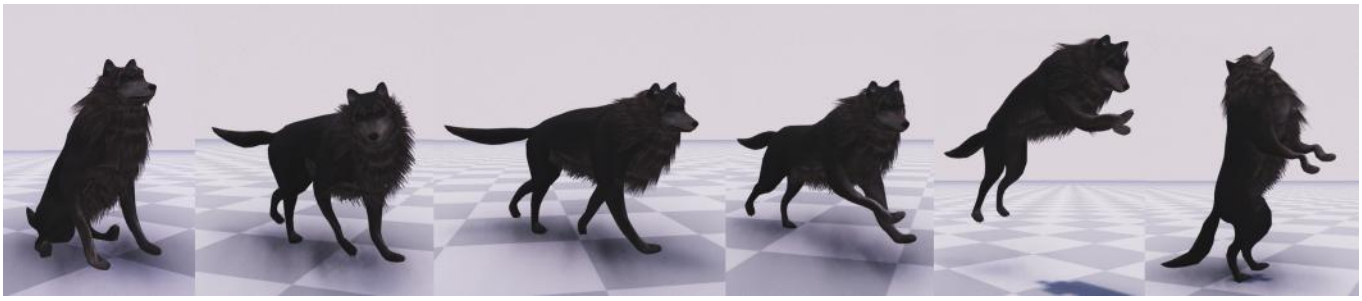

Fig. 1. A selection of results using our method for quadruped animation. We show some different modes for sitting, turning trot, pace, canter, jumping and standing from left to right. The locomotion gaits are not labeled individually, but naturally produced by the movement velocity control.

Quadruped motion includes a wide variation of gaits such as walk, pace, trot and canter, and actions such as jumping, sitting, turning and idling. Applying existing data-driven character control frameworks to such data requires a significant amount of data preprocessing such as motion labeling and alignment. In this paper, we propose a novel neural network architecture called Mode-Adaptive Neural Networks for controlling quadruped characters. The system is composed of the motion prediction network and the gating network. At each frame, the motion prediction network computes the character state in the current frame given the state in the previous frame and the user-provided control signals. The gating network dynamically updates the weights of the motion prediction network by selecting and blending what we call the expert weights, each of which specializes in a particular movement. Due to the increased flexibility, the system can learn consistent expert weights across a wide range of non-periodic/periodic actions, from unstructured motion capture data, in an end-to-end fashion. In addition, the users are released from performing complex labeling of phases in different gaits. We show that this architecture is suitable for encoding the multi-modality of quadruped locomotion and synthesizing responsive motion in real-time.

CCS Concepts: • **Computing methodologies → Motion capture**; *Neural networks*;

Additional Key Words and Phrases: neural networks, locomotion, human motion, character animation, character control, deep learning

ACM Reference Format:
He Zhang†, Sebastian Starke†, Taku Komura, and Jun Saito. 2018. Mode-Adaptive Neural Networks for Quadruped Motion Control. *ACM Trans. Graph.* 37, 4, Article 145 (August 2018), 11 pages. https://doi.org/10.1145/3197517.3201366

1 INTRODUCTION

Quadruped animation is one of the unsolved key problems in computer animation. It has particular relevance for applications in computer games and films, and also presents a challenging topic in robotics. When animating quadrupeds, animators must go through special training to design a wide range of complex movements. This complexity is inherently from the multi-modality of quadruped motions. For example, there are a number of locomotion modes [1] including walk, pace, trot, canter, and gallop, where the movements and the phases of the torso and limbs vary in a complex manner (see Fig. 2).

To the best of our knowledge, there is no prior work on systematically constructing quadruped motion controllers in a data-driven fashion. This difficulty stems from the complexity mentioned above, along with the issue that quadruped animals cannot be directed like humans for a controlled data acquisition. As a result, the captured data is often less structured with a wide range of random actions performed one after another. When designing character controllers using such data, engineers need to manually/semi-automatically extract gait cycles and transitions from the data, stitch them together, and tune parameters of motion trees and finite state machines.

† He Zhang and Sebastian Starke are joint first authors.

Authors' addresses: He Zhang†, University of Edinburgh, s1676928@sms.ed.ac.uk; Sebastian Starke†, University of Edinburgh, sebastian.starke@ed.ac.uk; Taku Komura, University of Edinburgh, tkomura@ed.ac.uk; Jun Saito, Adobe Research, jsaito@adobe.com.

0730-0301/2018/8-ART145 $15.00
https://doi.org/10.1145/3197517.3201366

[1] Locomotion modes is the term to represent the variation in movements the animals use to propel, here which is defined based on the binary footfall patterns along the timeline that the end effectors contact the ground. "Locomotion modes" and "gait types" are interchangably used in this article.

Motion control using neural networks has recently demonstrated success in producing high-quality animation for biped locomotion with clear cycles [Holden et al. 2017]. However, merely applying the same framework to quadrupeds fails because defining single phase for all four legs is not possible in the transition between gaits with very distinct footfall patterns. This also makes manual phase labeling of unstructured quadruped motion data with complex gait transitions impractical.

In this paper, we propose a novel network architecture called *Mode-Adaptive Neural Networks (MANN)* that can learn a locomotion controller from a large amount of unstructured quadruped motion capture data. The system is composed of the motion prediction network and the gating network. At each frame, the motion prediction network computes the character state in the current frame given the state in the previous frame and the user-provided control signals. The gating network dynamically updates the weights of the motion prediction network by selecting and blending what we call the expert weights, each of which specializes in a particular movement. This architecture provides flexibility such that the system can learn consistent features across a wide range of non-periodic actions and periodic unlabeled gait types. This framework can release the developers from the tedious and difficult process of phase labeling, where the unstructured quadruped motion capture data of different gait types must be aligned along the timeline. In particular, our model does not require individual labels for different gaits which are often difficult to distinguish even for humans, and thus avoids gait mislabeling during data preprocessing.

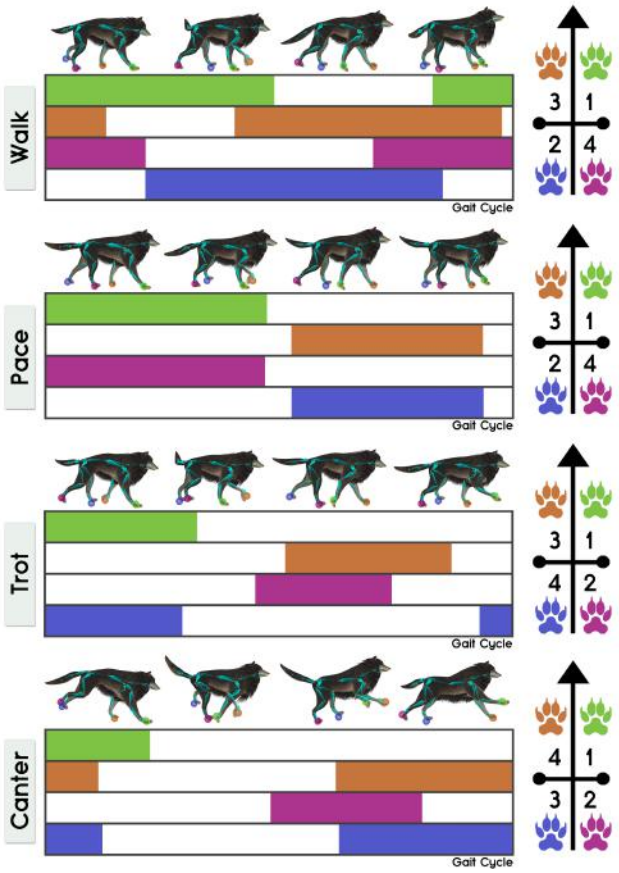

Fig. 2. Footfall patterns for different quadruped locomotion modes generated by referring to [Huang et al. 2013]. The horizontal bars indicate the stance phase in each leg. The image in the right indicates the sequence of the foot contact during the gait cycle.

The contributions of the paper can be summarized as follows:

- The first systematic approach for constructing data-driven quadruped character controllers that can synthesize animations in production-quality with a wide variety of locomotion modes and transitions between them.
- A novel end-to-end neural network architecture that can learn from unstructured quadruped motion capture data without providing labels of the phase and locomotion gaits.
- A comprehensive evaluation of the proposed architecture through comparison with existing approaches.

2 RELATED WORK

Quadruped motion synthesis has mostly been done through procedural modeling and physics-based control. Procedural animation is useful for animating virtual creatures with many legs [Hecker et al. 2008] or hexapoda [Karim et al. 2013]. Kry et al. [2009] animate quadrupeds using modal analysis. Although these methods can produce interesting stable gait cycles, they have difficulty in producing subtle, realistic movements when responding to quick user inputs, where physical rules such as the conservation of momentum and ground reaction forces strongly influence the motions. Simulating such movements requires either using physically-based animation or data-driven techniques.

In the rest of this section, we first review physics-based quadruped control, and then data-driven techniques which can potentially be applied to quadruped motion synthesis. Finally, we briefly review mixture of experts, which is a concept that inspires our approach.

2.1 Physics-Based Quadruped Controllers

Physically-based controllers are effective to generate dynamic movements, where the animals make use of elasticity, energy minimization and conservation of momentum. Such methods can be divided into trajectory-based approaches where the motion is optimized based on physical properties such as torques, momentum and feasibility [Levine and Popović 2012; Wampler and Popović 2009; Wampler et al. 2014], and torque-based approaches where the body is directly driven by torques [Coros et al. 2011; Liu and Hodgins 2017; Peng et al. 2015, 2016; Raibert and Hodgins 1991; van de Panne 1996].

Trajectory-Based Approaches: Many trajectory optimization techniques are developed for human motion synthesis [Al Borno et al. 2013; Liu et al. 2005; Liu and Popović' 2002; Ye and Liu 2012] and some of these ideas are applied for quadruped motion synthesis. Wampler and Popovic [2009] compute the motion of various types of animals including quadrupeds by optimizing an objective function composed of the torques and constraint terms over cyclic gaits. Wampler et al. [2014] extend this technique to predict the motion

of animals whose appearance is known but the motion is unknown. These are spacetime approaches which cannot be applied for online applications. Levine and Popovic [2012] adapt the dog locomotion to dynamically deforming terrains in a quasi-physical manner. Such an approach is useful for adapting a small number of exemplar motions to different environments, but still need a set of motion to start from, which leads to data-driven approaches that are discussed later.

Torque-Based Approaches: Torque-based controllers have long been developed for human characters [Liu and Hodgins 2017; Peng et al. 2016; Yin et al. 2007] and quadruped characters[Coros et al. 2011; Peng et al. 2015; Raibert and Hodgins 1991; van de Panne 1996]. Designing such controllers is not trivial as the characters should perform the desired motion while keeping balance. Raibert and Hodgins [1991] develop control strategies for trotting, bouncing and galloping gaits. Van de Panne [1996] optimizes the parameters of a control graph to control a cat model using speed as the primary metric. Coros et al. [2011] design a detailed controller for quadrupeds that can simulate a wide range of locomotion including walk, trot, canter and traverse gallop by optimizing PD controllers to follow reference movements from videos while satisfying constraints. Peng et al. [2015] apply reinforcement learning to control quadruped models to adapt to different terrains in 2D. Designing a stable controller in a high dimensional state space is a very difficult problem and thus low-dimensional features need to be carefully hand-tuned to keep the controller stable. Peng et al. [2016] overcome this problem by applying deep reinforcement learning, where the feature space is automatically computed from the data. This approach was further enhanced for 3D environments and applied to biped characters in [Peng et al. 2016].

Physically-based controllers are very powerful tools for designing dynamic plausible movements though some subtle minor voluntary movements that make the motion realistic tend to be skipped due to the difficulty in describing them from simple rewards such as moving forward, energy minimization and balance control. Adversarial training as done in [Merel et al. 2017a] could be a future direction to overcome this issue.

2.2 Data-Driven Character Control

A counterpart of physically-based animation is data-driven character animation techniques that make use of motion capture data for interactive character control. Data structures such as motion graphs [Arikan and Forsyth 2002; Kovar et al. 2002; Lee et al. 2002] are introduced to synthesize continuous character movements from unstructured motion capture data. As the connectivity of the graph can significantly affect the responsiveness of the controlled character, computer games and other interactive applications often use a more straightforward structure such as finite state machines where the connectivity is explicit and the subsequent motion is predictable, as proposed by Lau and Kuffner [2005].

In the rest of this section, we first review methods that make use of early machine learning approaches for motion synthesis, time series models which model the motion from the dynamic perspective and then the recent methods that make use of deep learning-based approaches.

Motion Synthesis by Classic Machine Learning Techniques: To improve the transitions between the actions and increase the generality of the output motion, the system should preferably be synthesizing novel motions rather than simply replaying the given data. Various techniques based on machine learning such as K-Nearest Neighbours (KNN), principal component analysis [Chai and Hodgins 2005; Min and Chai 2012; Safonova et al. 2004; Tautges et al. 2011], radial basis functions (RBF) [Kovar and Gleicher 2004; Rose et al. 1998], reinforcement learning [Safonova and Hodgins 2007] and Gaussian processes (GP) [Grochow et al. 2004; Ikemoto et al. 2009; Mukai and Kuriyama 2005] are studies to achieve this goal.

Most methods based on classic machine learning techniques suffer from scalability issues: they first require a huge amount of data preprocessing including motion classification and alignment. KNN requires keeping all the motion data, and kernel-based approaches like RBF and GP require square order memory and cube order computation time. PCA-based approaches perform well in high-dimensional problems such as when motion trajectories are used as the representation, but requires local structures for time-series models [Chai and Hodgins 2005; Tautges et al. 2011], where again a lot of preprocessing is needed.

Time Series Models: Time series models predict the motion for the current frame given the motion in the previous frames. Such models are useful for real-time applications such as computer games. Similar to motion interpolation, methods based on linear regression [Hsu et al. 2005; Xia et al. 2015], KNN [Lee et al. 2010] and kernel-based approaches [Wang et al. 2008] are proposed. Linear models [Hsu et al. 2005] are simple but have issues with modeling nonlinearity. Xia et al. [2015] cope with this problem by using expert gates, though their method requires predefining the classes of each experts, which we overcome in our research. Kernel-based approaches [Levine et al. 2012; Wang et al. 2008] can model nonlinearity in motion, but the trained model is limited to express a specific type of motion. Motion fields [Lee et al. 2010] require preserving the original data and conducting KNN search during runtime, which limits the scalability.

Learning Character Motion by Neural Networks: Neural networks are getting attention due to their scalability and high run-time performance. Neural networks can learn from a massive amount of data, while keeping their sizes much smaller than the original data sizes. Taylor et al. [2009; 2011] present human motion capture data can be learned by conditional Restricted Boltzmann Machines (cRBM), although it suffers from the noise due to per-frame sampling. Recurrent Neural Networks (RNN) that predict the following posture from the previous motion are attractive frameworks for learning time series data such as human motion, but are known to suffer from converging to an average pose [Fragkiadaki et al. 2015]. Li et al. [2017] avoid the problem by linking the network predictions into future inputs, which drags the generated motion of the RNN back to the original motion data. The focus of these works are in the reconstruction of the time series data, and they do not receive extra control signals for the control. How they interpolate between different movements that follow the user instructions is yet to be examined.

For the character animation purpose, Holden et al. [2015] use convolutional neural network (CNN) and learn temporal filters as the

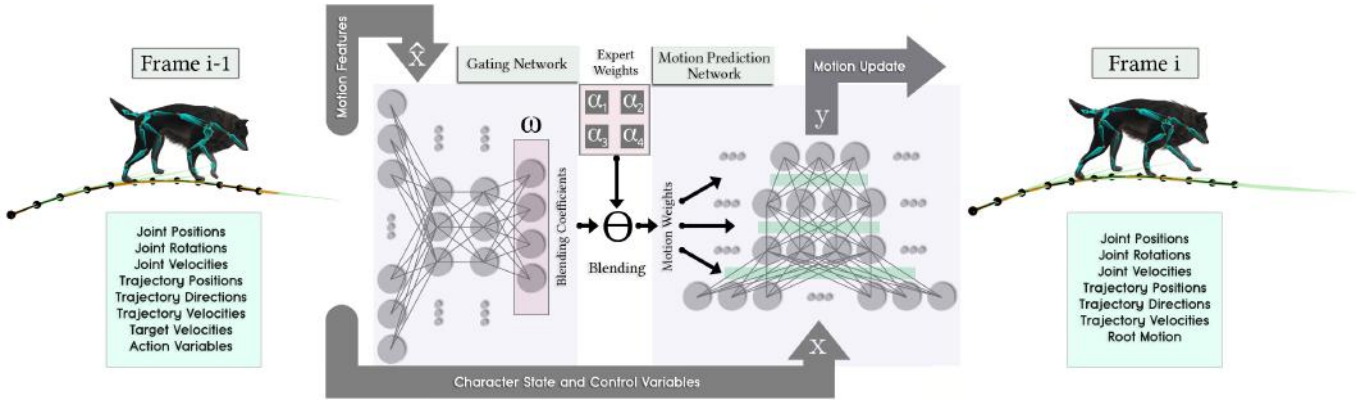

Fig. 3. The architecture of our neural network composed of the gating network and the motion prediction network. The gating network takes as input the current end effector velocities of the feet, the desired velocity and the action vector. The motion prediction network takes as input the posture and trajectory control variables from the previous frame, and predicts the updated posture and trajectory for the current frame.

representation of motions. They also demonstrate CNN's capabilities to map high-level user instructions to character movements [Holden et al. 2016]. Holden et al. [2017] propose a real-time framework, the phase-functioned neural network (PFNN), that maps the inputs from a gamepad controller to the motion of the character. Naive application of the same technique to quadrupeds exposes issues caused by the complexity of the quadruped motion and the difficulty in gait type and phase labeling. Most importantly, they introduce the phase as a parameter to align the locomotion along the timeline, such that they do not blend motion at the wrong timing. This is only possible if the way the feet contact the ground is consistent. For quadrupeds, this pattern can drastically change between modes, and thus can result in artifacts. We present such examples in Section 8.

2.3 Mixture of Experts

Mixture of Experts (MoE) [Jacobs et al. 1991; Jordan and Jacobs 1994] is a classic machine learning approach where a number of experts are used to cope with the inputs in different regions. A gating network decides which set of experts to use for the given input. Once trained, the experts become locally specialized for some subdomain assigned by the gating network. We refer the readers to Yukel et al. [2012] for a broad survey. Recently its combination with deep learning architectures [Chang et al. 2018; Eigen et al. 2013; Shazeer et al. 2017] is showing promising results. Our architecture has similarity with MoE, as we also use a gating function, though our blending is occurring at the feature level, while for MoE it happens at the output level.

3 SYSTEM OVERVIEW

Our system is a time series model that predicts $\mathbf{y}$, the state of the character in the current frame, given $\mathbf{x}$, the state in the previous frame and the user control signals. For achieving this task for characters such as quadrupeds who produce a wide range of periodic and non-periodic movements, we propose a novel neural network structure called Mode-Adaptive Neural Networks (MANN) (see Section 6, Fig. 3). The motion in the current frame is computed by a network that we call the motion prediction network (see Section 6.1, Fig. 3, right), whose network weights are dynamically computed by another neural network structure that we call the gating network (see Section 6.2, Fig. 3, left). The gating network receives a motion feature $\hat{\mathbf{x}}$, which is a subset of $\mathbf{x}$, and computes the blending coefficients of the expert weights (see Fig. 3, center), each of which is trained to be specialized in particular movements.

To prepare the training dataset, we first preprocess the dog motion capture dataset and add the action labels to the data (see Section 4), and prepare the input and output vectors (see Section 5). During training, the entire network is trained end-to-end using the prepared training dataset (see Section 7). During runtime, the system animates the character in real-time using the previous state of the character and the control signals provided by the user (see Section 8).

4 DATA PREPARATION

Here we describe the motion capture and motion classification stages, and present the breakdown of our data.

Dog Motion Capture. Our motion capture data consists of 30 minutes of unstructured dog motion capture data that is composed of various locomotion modes including walk, pace, trot, and canter, as well as other types of motions such as sitting, standing, idling, lying and jumping. The size of the data is doubled by mirroring. All the motion data was only captured on a flat terrain due to the limitation of the capture facilities. A skeleton model that is composed of 27

Fig. 4. The skeleton structure of the dog model used in our experiments. It is composed of 27 bones and has 81 degrees of freedom in total.

Table 1. The breakdown of our dog motion dataset for training. This dataset includes the original and mirrored unstructured dog motion capture.

Motion Type	time (sec)	frames	ratio (%)
idle	1614.37	96862	36.42
locomotion	1828.70	109722	41.25
jump	30.27	1816	0.68
sit	497.93	29876	11.23
lie	397.13	23828	8.96
stand	64.83	3890	1.46

bones (see Fig. 4) is fitted to the captured data. The body has 81 degrees of freedom in total.

Motion Classification. The motions are first classified into locomotion, sitting, standing, idling, lying and jumping such that the user can control the character during runtime by specifying such labels. This process is done manually, though, it should not be difficult to automate this process, as the motions are rather distinct between these classes. The breakdown of the ratio of the motion data is shown in Table 1.

Locomotion Modes. In our paper, we specifically work on four types of locomotion modes: walk, pace, trot, and canter. Although our system does not require the labels of the locomotion modes for controlling the character during runtime, we analyzed the distribution of the modes in the dataset. Classification of the data into locomotion modes is a rather difficult process due to the complex transitions and the ambiguous cases; we first roughly classify the motions based on the velocity profile and then manually divide them into each mode. The final breakdown/correlation of the speed and the mode types are visualized in Fig. 5. This correlation matches well to the model by Coros et al. [2011].

5 INPUT AND OUTPUT FORMATS

The input and output data formats of our system follow those of Motion Matching [Clavet 2016] and PFNN [Holden et al. 2017]. The data is composed of the ground trajectory transformations and velocities in the past and future states, the labels of the action type at those frames, and the body joint transformations and velocities of

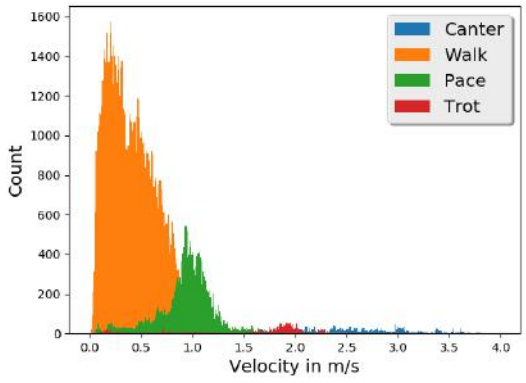

Fig. 5. Distribution of velocities for different quadruped locomotion modes.

the current state. More specifically, for each frame i, we uniformly sample another 11 frames which cover one second in the future and past. From these 12 frames of motion, we extract features including the character positions, rotations and velocities relative to the body's local transformation, which is projected onto the ground with its rotation facing in the character's forward direction. We call this the root transformation of our system, which also represents the trajectory transformation for the current frame. We then include the action labels at those 12 frames to be part of the state vector. Finally, all trajectory and body joint transformations are computed relative to the root trajectory transformation. The desired velocity of each frame is simply calculated by the length of the future trajectory at this particular state.

The input vector at frame i is $\mathbf{x}_i = \{ \mathbf{t}_i^p \, \mathbf{t}_i^d \, \mathbf{t}_i^v \, \mathbf{t}_i^{\hat{v}} \mathbf{t}_i^a \, \mathbf{j}_{i-1}^p \, \mathbf{j}_{i-1}^r \, \mathbf{j}_{i-1}^v \} \in \mathbb{R}^n$, where $\mathbf{t}_i^p \in \mathbb{R}^{2t}$ are user-predicted trajectory positions of t samples (in our case 12) in a 2D-horizontal plane relative to the current state i, $\mathbf{t}_i^d \in \mathbb{R}^{2t}$ are trajectory forward facing directions relative to state i, $\mathbf{t}_i^v \in \mathbb{R}^{2t}$ are the trajectory velocities relative to state i, $\mathbf{t}_i^{\hat{v}} \in \mathbb{R}^{1t}$ are the desired trajectory velocity values in state i, and $\mathbf{t}_i^a \in \mathbb{R}^{6t}$ are one-hot vectors for the character action type of the samples along the trajectory. $\mathbf{j}_{i-1}^p \in \mathbb{R}^{3j}$, $\mathbf{j}_{i-1}^r \in \mathbb{R}^{6j}$ and $\mathbf{j}_{i-1}^v \in \mathbb{R}^{3j}$ are the relative joint positions, rotations and velocities of the previous state where j denotes the number of joints (in our case 27). In addition to the PFNN [Holden et al. 2017], also using the joint rotations for the input of the network instead of only for the output layer could be observed to immediately produce a much sharper motion for our quadruped motion dataset.

Similarly, the output is $\mathbf{y}_i = \{ \mathbf{t}_{i+1}^p \, \mathbf{t}_{i+1}^d \, \mathbf{t}_{i+1}^v \, \mathbf{j}_i^p \, \mathbf{j}_i^r \, \mathbf{j}_i^v \, \dot{\mathbf{r}}_i^x \, \dot{\mathbf{r}}_i^z \, \dot{\mathbf{r}}_i^a \} \in \mathbb{R}^m$ where $\mathbf{t}_{i+1}^p \in \mathbb{R}^{2t}$, $\mathbf{t}_{i+1}^d \in \mathbb{R}^{2t}$ and $\mathbf{t}_{i+1}^v \in \mathbb{R}^{2t}$ are the predicted relative trajectory positions, directions and velocities for the next state, and $\mathbf{j}_i^p \in \mathbb{R}^{3j}$, $\mathbf{j}_i^r \in \mathbb{R}^{6j}$ and $\mathbf{j}_i^v \in \mathbb{R}^{3j}$ are the relative joint positions, rotations and velocities for the current state. $\dot{\mathbf{r}}_i^x \in \mathbb{R}$ and $\dot{\mathbf{r}}_i^z \in \mathbb{R}$ are the root translational x and z velocities relative to the previous frame which define the local transformation into the current frame, and $\dot{\mathbf{r}}_i^a \in \mathbb{R}$ is the corresponding root angular velocity in the 2D-horizontal plane. This is similar to the data preparation of PFNN, though our system does not output any information about the foot contact or the phase increment, but additionally includes the trajectory velocities to get a better controllability. Also, we represent the joint rotations by the relative forward and upward vectors in order to avoid quaternion interpolation issues by the neural network during training. Those are obtained from the original quaternions in the motion capture, and transformed back for generating the motion during runtime.

6 MODE-ADAPTIVE NEURAL NETWORKS FOR QUADRUPED CONTROL

In this section, we describe the architecture of the Mode-Adaptive Neural Networks (MANN) in Fig. 3, which is composed of the motion prediction network and the gating network. We first describe the motion prediction network, which predicts the motion in the current frame given the state in the previous frame, and then describe the gating network, which computes the weights of the motion prediction network dynamically.

6.1 Motion Prediction Network

The motion prediction network, whose operation is denoted by $\mathbf{\Theta}(\cdot)$, is a simple three layer neural network that receives the previous state of the character $\mathbf{x}$, and outputs the data in the format of $\mathbf{y}$ as follows:

$$\mathbf{\Theta}(\mathbf{x};\boldsymbol{\alpha}) = \mathbf{W}_2\ \mathrm{ELU}(\ \mathbf{W}_1\ \mathrm{ELU}(\ \mathbf{W}_0\ \mathbf{x} + \mathbf{b}_0) + \mathbf{b}_1) + \mathbf{b}_2, \quad (1)$$

where the parameters of the network $\boldsymbol{\alpha}$ are defined by $\boldsymbol{\alpha} = \{\mathbf{W_0} \in \mathbb{R}^{h\times n}, \mathbf{W_1} \in \mathbb{R}^{h\times h}, \mathbf{W_2} \in \mathbb{R}^{m\times h}, \mathbf{b_0} \in \mathbb{R}^{h}, \mathbf{b_1} \in \mathbb{R}^{h}, \mathbf{b_2} \in \mathbb{R}^{m}\}$. Here h is the number of units used in the hidden layers, which in our work is set to 512, and the activation function used is the exponential rectified linear function [Clevert et al. 2015] defined by

$$\mathrm{ELU}(x) = \max(x, 0) + \exp(\min(x, 0)) - 1. \quad (2)$$

The neural network weights $\boldsymbol{\alpha}$ is computed by blending K expert weights $\boldsymbol{\beta} = \{\boldsymbol{\alpha}_1, ..., \boldsymbol{\alpha}_K\}$, each of which in the form of neural network weight configurations: $\boldsymbol{\alpha} = \sum_{i=1}^{K} \omega_i \boldsymbol{\alpha}_i$. K is a meta parameter that can be adjusted according to the complexity and size of the training data, and $\omega = \{\omega_1, ..., \omega_K\}$ are the blending coefficients computed by gating network described in Section 6.2. We found that $K = 4$ is enough to generate high-quality motions, though $K = 8$ is able to produce even better results with sharper movements. We will discuss about this in Section 8.

6.2 Gating Network:

The gating network, whose operation is denoted by $\Omega(\cdot)$, is a three layer neural network that computes the blending coefficients ω given the input data $\mathbf{x}$:

$$\Omega(\hat{\mathbf{x}};\mu) = \sigma(\mathbf{W}'_2\ \mathrm{ELU}(\ \mathbf{W}'_1\ \mathrm{ELU}(\ \mathbf{W}'_0\ \hat{\mathbf{x}} + \mathbf{b}'_0) + \mathbf{b}'_1) + \mathbf{b}'_2), \quad (3)$$

where $\hat{\mathbf{x}} \in \mathbb{R}^{19}$ is a subset of $\mathbf{x}$ that are the foot end effector velocities, the current action variables and the desired velocity of the character. The parameters of the network μ are defined by $\mu = \{\mathbf{W}'_0 \in \mathbb{R}^{h'\times 19}, \mathbf{W}'_1 \in \mathbb{R}^{h'\times h'}, \mathbf{W}'_2 \in \mathbb{R}^{K\times h'}, \mathbf{b}'_0 \in \mathbb{R}^{h'}, \mathbf{b}'_1 \in \mathbb{R}^{h'}, \mathbf{b}'_2 \in \mathbb{R}^{K}\}$ where h' is the number of hidden layer units which is set to 32. $\sigma(\cdot)$ is a softmax operator that normalizes the inputs such that they sum up to 1, which is required for the further linear blending.

We have tested various other inputs including the full $\mathbf{x}$, the end effector positions, and the combination of their position and velocity, as well as the rotations, and found those features alone to clearly produce the best results. This could be due to the strong correlation of the feet velocity with the locomotion phase, resulting in an effect similar to the phase function in [Holden et al. 2017], which helps to avoid blending between motions of different phases. Ideally, the network could learn the informative features from the input, though this could be difficult due to our relatively small amount of training data as listed in our experiments in Section 8. Meanwhile, we observed that the using the action variables and the desired velocity helps to improve the controllability and responsiveness of the character.

7 TRAINING

The entire network is trained end-to-end using the processed motion capture data. The input $\mathbf{x}$ and $\mathbf{y}$ for each frames are stacked into matrix form: $\mathbf{X} = [\mathbf{x}_1\mathbf{x}_2...], \mathbf{Y} = [\mathbf{y}_1\mathbf{y}_2...]$. These values are shifted and scaled using their mean and standard deviation such that the data is normalized. As the cycles of trot and canter are very limited, we copied the data of those motions 11 times each, which helps them to appear during runtime more robustly.

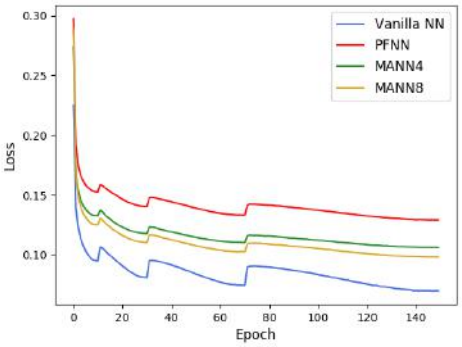

Fig. 6. Learning curves of vanilla neural network, PFNN and MANN with 4 or 8 expert weights respectively. The warm restarts of the AdamWR algorithm at epoch 11, 31 and 71 result in hikes of the loss.

The goal of training our network is that for a given set of inputs $\mathbf{X}$, we can produce the corresponding output variables $\mathbf{Y}$, which is a typical regression task with the following cost function, which is the mean squared error between the predicted output and the ground truth:

$$\mathrm{Cost}(\mathbf{X}, \mathbf{Y}; \boldsymbol{\beta}, \mu) = \|\mathbf{Y} - \mathbf{\Theta}(\mathbf{X}, \Omega(\hat{\mathbf{X}}; \mu); \boldsymbol{\beta})\|_2^2, \quad (4)$$

We use the stochastic gradient descent algorithm with the warm restart technique of AdamWR [Loshchilov and Hutter 2017], which automatically calculates the derivatives of the cost function with respect to $\boldsymbol{\beta}$ and μ. The model is implemented in Tensorflow [Abadi et al. 2016]. As AdamWR does the regularization within the optimization procedure, no regularization term is contained in the cost function. Instead, it uses two parameters T_i and T_{mult} that control the decrease and restart of the learning rate η and weight decay rate λ, which we chose to be initialized as $\eta = 1.0 \cdot 10^{-4}$ and $\lambda = 2.5 \cdot 10^{-3}$. T_i is the total number of epochs within the i-th run/restart with an initial value of 10. At every restart, we choose T_i to be multiplied by the factor of $T_{mult} = 2$, i.e. $T_{i+1} = 2T_i$. We set the total epochs as 150, hence a total of 3 restarts will occur at epoch 11, 31 and 71. During the training, we iterate over the data in mini-batches with a size of 32, where the training samples in each mini-batch are selected randomly. Dropout is applied with a retention probability of 0.7. Full training with 4 or 8 experts networks takes around 20 or 30 hours on a NVIDIA GeForce GTX 970 GPU, respectively. In Fig. 6, we show the learning curves of different methods. Note that a lower training loss or test loss does not necessarily correspond to a higher quality of motion.

8 EXPERIMENTS AND RESULTS

In this section, we first describe the character control scheme during runtime, and then show the results when controlling the character interactively during runtime.

Runtime Control: The system is implemented in the Unity 3D engine, and requires ∼2 ms per frame for running the neural network single-threaded on an Intel Core i-7 CPU using a port of the Eigen library. The memory usage for the trained model is ∼22 MB using 8 expert weights. The character can be controlled by discrete and continuous control signals. By selecting the key that corresponds to the element of the one hot vector, the desired action, which is either of sit, stand, idle, lie, jump and move, is launched. The speed of the character, which is also included in the control parameters, is discretely specified by the arrow keys, increasing and interpolating the velocity in the forward, backward, left and right directions. The facing direction of the locomotion is controlled by two keys in order to perform a turning motion. Both the target velocity and direction are smoothly interpolated and finally used to predict the future trajectory which the user wants to navigate. The curve of the trajectory is extrapolated using an exponentially-weighted bias value which defines the maximum length of the future trajectory — and thus providing the desired velocity of the character in m/s. This always results in a smooth trajectory, which is crucial for the input of the neural network to achieve good looking motion by trying to resemble realistic trajectories as in the original motion capture data.

Runtime Results: Various gaits including walk, pace and canter are successfully produced by setting the speed to $0.5m/s$ (walk), $1.1m/s$ (pace), $1.9m/s$ (trot) and $3.3m/s$ (canter), respectively. Smooth transitions can be produced by gradually changing the speed. All types of locomotion quickly respond to the direction and velocity input, although some gaits do not include the turning motion. This means the network is well generalizing the turning motion of the standard gait like walk to other gaits. Motions such as sit, lie, stand and jump are successfully launched with little delay when the corresponding hotkey is pressed (see Fig. 1). This indicates that the network can learn at which state particular actions may be performed. The readers are referred to the supplementary video for further details.

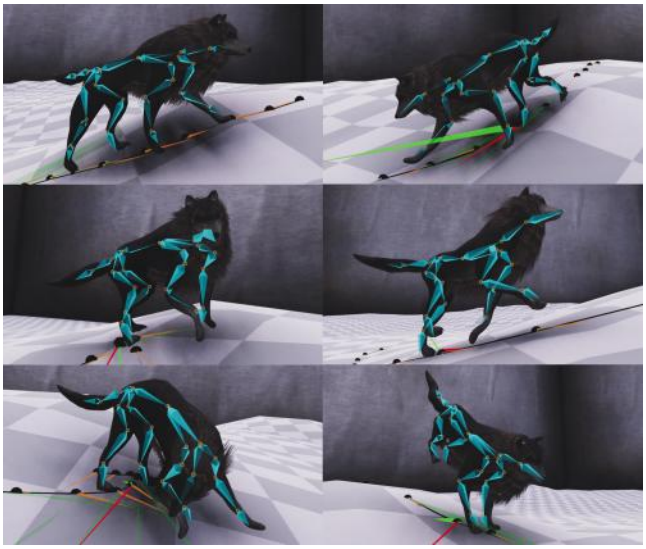

Fig. 7. The quadruped character walking over a smooth uneven terrain synthesized by full-body inverse kinematics.

Finally, we prepare an environment where the character can move along an uneven terrain (see Fig. 7). Instead of training the character to walk over different terrains as done in [Holden et al. 2017], we simply apply a CCD full-body inverse kinematics to the character movements. The positions of the paw end effectors are post-processed such that they incorporate the terrain height offset in addition to the generated motion, and the spine joints are simultaneously updated with respect to the surrounding heights. This is required to avoid outstretched limbs when walking up- and downhills, and thus to create more natural movements.

9 EVALUATION

We evaluate our system through comparison to existing methods in terms of the motion quality, foot sliding artifacts, leg stiffness, and responsiveness. We also examine the activation of the expert weights, and analyze their functionality by deactivating them during the motion.

Comparison with Other Frameworks: We compare the performance of the proposed system with baselines such as vanilla feedforward neural networks and phase-functioned neural networks (PFNN) with semi-automatic phase labeling. Note that all the rest of the inputs and outputs are kept the same as our method, and no foot contact information are included in the state vector of each method. For a fair comparison, we make the number of layers and parameters in the vanilla neural network and the PFNN the same as for the MANN using 4 expert weights. Thus, the vanilla network has 2048 units in the hidden layers, and the PFNN has 4 control points, each of which has 512 units in the hidden layers (4×512). Meanwhile, we also show the results of MANN with 8 expert weights, and list the ground truth (GT) values from the original motion capture data.

Vanilla feedforward consistently suffers from undesired blending which results in blurry movements and loss of high-frequency components. This artifact is apparent even for a constant walk and pace cycles, for which the data is relatively rich. The artifacts such as foot skating and stiff legs become significantly worse when conducting turning behaviors and when landing after jumps.

In comparison to PFNN, our system performs significantly better, despite the fact that ours does not require any phase labels. For testing the PFNN, the phase labels are added to the dog motion data. The phases are defined based on the velocity profile of the front two legs, computed and aligned by an optimization technique [2]. The gait type is specified through one-hot vectors. Such a PFNN performs well in general, though the motion of the back legs often appear stiff and unnatural, and also resulting in foot skating. This can be due to the difficulty of defining a consistent phase function that can well align different gait types; as we define the phase based on the velocity of the front legs, the artifacts become more observable at the back legs.

[2]The cost function is defined by fitting the parameters of a trigonometric function such that it matches the velocity update of the feet. This takes into account symmetry information in the motion, and adaptively detects different types of locomotion. The beginning and end of each cycle is then analytically extracted by computing the either negative or positive turning points. This enables us to automatically label the phase of the whole motion dataset with high accuracy.

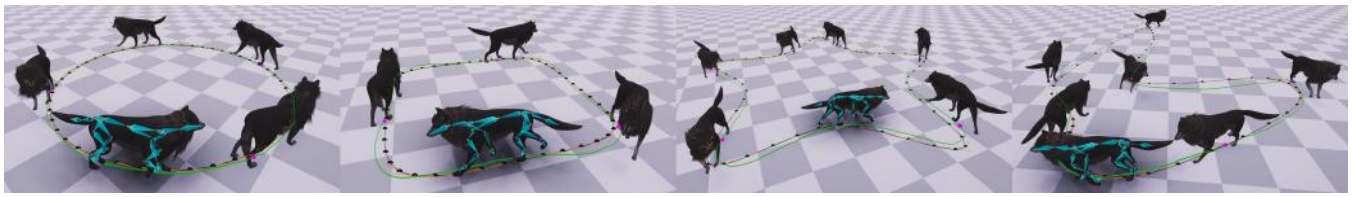

Fig. 8. The character following predefined trajectories on the ground. The parameter τ for correction with the predicted trajectories of the network is set to 0.5 in order to perform realistic motions along artificial paths. It can be observed that the character is following the curves well even when there are sudden turns.

Table 2. The average foot skating for all legs and the back legs in the ground truth data, and when using the vanilla NN, PFNN and MANN models with 4 or 8 expert weights respectively.

Motion	GT	vNN	PFNN	MANN4	MANN8	cm/frame
Walk	0.21	0.32	0.25	0.23	0.22	all legs
	0.08	0.15	0.12	0.10	0.09	back legs
Canter	0.14	0.24	0.19	0.17	0.16	all legs
	0.05	0.19	0.11	0.08	0.06	back legs
Turn	0.17	0.28	0.34	0.24	0.20	all legs
	0.06	0.17	0.21	0.13	0.08	back legs

In biped locomotion, it is relatively easy to align the motion based on the foot contact pattern for different modes like walking and running. However, such a simple rule is difficult to be adopted for quadrupeds, where the relative phases of the swing and support legs change significantly across the locomotion modes. On the contrary, the gating network has more degrees of freedom for blending the features to minimize the cost function, which offers more flexibility to align different modes of locomotion.

Foot Skating Artifacts: We also compare our system with the vanilla neural networks and PFNN in terms of foot skating artifacts, listed in Table 2. The foot velocities are added if a position height h is within a maximum threshold of $H = 2.5$cm. Each velocity magnitude v in the horizontal plane is further weighted by an exponential interpolation $s = v(2 - 2^{\frac{h}{H}})$ to estimate the amount of skating s during motion, where the exponent is clamped between 0 and 1.

Leg Stiffness: In PFNN, the foot skating is caused by the back leg stiffness, which happens due to the blending of back leg movements

Table 3. The average angular update per joint along all legs and the back legs in the ground truth data, and when using the vanilla NN, PFNN and MANN models with 4 or 8 expert weights respectively.

Motion	GT	vNN	PFNN	MANN4	MANN8	°/frame
Walk	3.87	3.02	3.36	3.43	3.69	all legs
	3.16	2.47	2.71	2.82	3.05	back legs
Canter	6.53	4.61	4.63	5.23	5.72	all legs
	5.56	3.73	3.52	4.56	5.14	back legs
Turn	2.74	1.37	1.84	2.24	2.31	all legs
	2.15	1.02	1.06	1.54	1.82	back legs

Table 4. Average values for position and angle deviation while aiming to smoothly follow predefined trajectories of different curves, when using vanilla NN, PFNN and MANN models.

Path	vNN	PFNN	MANN	
Circle	2.28	1.96	2.98	Position Deviation (**cm**)
	3.07	1.70	3.21	Angle Deviation (°)
Square	5.80	3.72	4.65	Position Deviation (**cm**)
	5.20	3.78	7.21	Angle Deviation (°)
Star	7.45	8.37	6.61	Position Deviation (**cm**)
	8.80	11.94	8.76	Angle Deviation (°)
Custom	8.14	9.50	5.31	Position Deviation (**cm**)
	7.69	9.28	6.75	Angle Deviation (°)

under a phase computed by the front leg movements. To quantitatively evaluate such stiffness, we compute the average update per joint angle along all legs and the back legs (see Table 3). It can be observed that the leg motion is much smaller in vanilla NN and PFNN compared to MANN, especially for the back legs.

Responsiveness: We finally evaluate the responsiveness and the path-following accuracy of our method. We create several predefined paths and instruct the character to follow them in different modes. In order to make the character follow the trajectory, we use a blending parameter τ which interpolates between the future points of the desired trajectory T^* as well as the corrected trajectory T^+ in the output of our network to obtain the input trajectory T for the motion update. This blending can be defined as $T = \tau T^* + (1-\tau)T^+$. The trajectories made by the character are shown in Fig. 8. We then measure the difference between the desired trajectory and the actual trajectory of the character considering their root transformations (see Table 4). It can be observed that the character is well following the paths and the average distance is low.

What are the Networks Learning?: To examine the situation, we plot the profile of the blending coefficients ω with $K = 4$ and 8 expert weights when the character is performing different types of actions and locomotion modes in Fig. 9. It can be observed that the weight values are equally cycling around for low speed movements (see Fig. 9, second bottom, bottom, upper rows), making the gating network function in the same fashion as the phase function in PFNN. The period that the expert weights corresponding to the purple line is active, becomes longer and constant as the speed of the locomotion increases, while the others tend to activate less in such situations. This means that these expert weights are trained to be responsible for fast movements, jumpy movements while the others could be

responsible for synthesizing rather slower movements. In particular, by selectively disabling single dimensions of blending coefficients, we observed the character no longer being able to perform certain movements, such as single types of locomotion modes, turning or jumping. Our observations for a learned network with eight trained blending coefficients are listed in Table 5. While most of them are clearly contributing to the overall quality of motion, a few became entirely responsible for specific actions or modes. Specifically for turning either left or right, it could be observed that turning into the opposite direction remained largely or even entirely unaffected.

Table 5. Resulting motion artifacts and disablings when selectively deactivating a weight α_i by ignoring the corresponding blending coefficient of the gating network. Some weights have learned features which are specifically responsible for certain motions.

α_i	Observation
1	*jumping motion fails (drops down after initiate)*
2	*left turning not possible (right turning works normally)*
3	*jumping motion fails; less accurate movements in general*
4	*right turning not possible (left turning works normally)*
5	*canter and trot motion not possible*
6	*only tiny movements along the limbs*
7	*right turning fails; everything else works*
8	*hard to activate modes; rather stiff limbs*

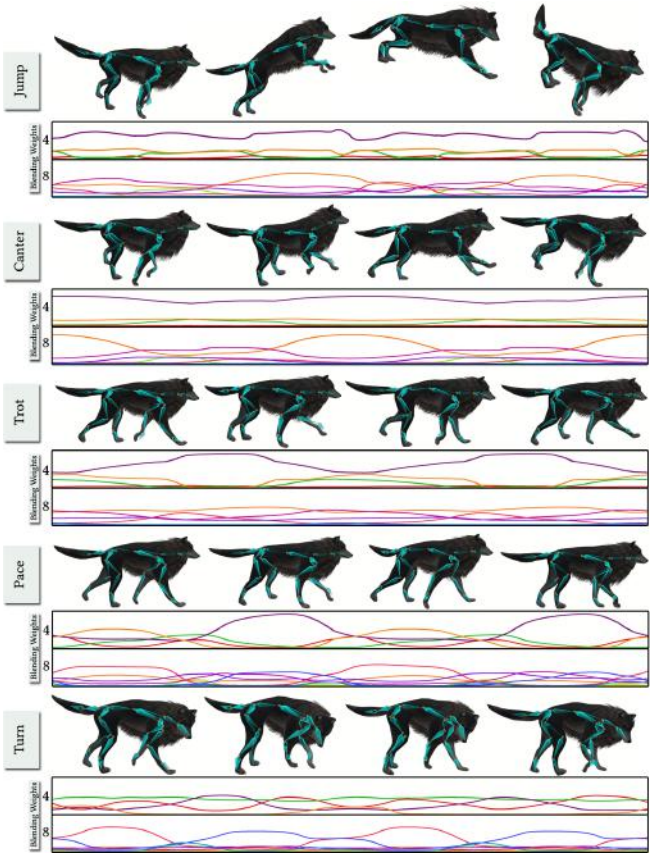

Fig. 9. The activation profile of the experts during different actions. The plots for two cycles of motions are shown.

Expert Imbalance: One concern that is reported in previous similar architectures [Bengio et al. 2015; Eigen et al. 2013; Shazeer et al. 2017] is that a small set of experts tend to receive higher weights by the gating network. This imbalance will increase as the favored experts will be trained even more rapidly. In fact, we observe this phenomenon when applying the original mixture of experts architecture [Jacobs et al. 1991] in our setup, where we prepare four experts and blend the outputs using a gating network: only one expert is trained well without exploiting others. Previous work in this area copes with this issue by imposing hard constraints [Eigen et al. 2013], or by imposing additional penalty in the loss that infavors such imbalance [Bengio et al. 2015; Eigen et al. 2013; Shazeer et al. 2017]. It seems such a penalty is not needed for our system with a small number of control points, as their usage becomes relatively balanced, and the system is performing well for animating the character.

10 DISCUSSIONS

Below we discuss the advantages of our proposed model for learning quadruped motion, followed by the limitations of our system.

Learning Time Series Models: Time series data are rather difficult to learn, often converging to average poses by blending poses at different phases in cyclic motions, or timing in acyclic motions. Also, the output motions tend to be smoothed out due to such averaging. Although a simple feedforward network can interpolate well where the samples are rich, such as during the gait cycle, the results appear smoothed out and unnatural with a lot of foot sliding when the transitions happen at configurations with less training data. PFNN is introduced to produce good interpolation even at such configurations by aligning the motion data along the phase dimension and only interpolating motion at the same phase. One way to view MANN is that it is a generalization of PFNN. The control points can be trained to be specialized for a motion at a certain timing and its activation can be further adjusted by the gating network in an unsupervised fashion. Once trained, bad interpolation can be reduced as there is little blending between different expert weights which are specialized for different phases.

Sparse Dataset: As mentioned above, deep neural networks can tend to fail at domains where there is less training data. The domain with sparse samples can be strongly affected by the surrounding domain with dense samples. The dog motion capture dataset could stay in such a distribution until the capture facilities become more lightweighted. One hour of motion capture data can be considered small for human data, but is relatively large for quadruped data. Given such nature of the training dataset, the proposed architecture is a good match to manage such locally sparse training data.

Limitations: Our dataset is limited to those from flat terrain and thus movements such as jumping on to/off from high positions

cannot be synthesized. The inverse kinematics approach that we presented is useful for simple motion synthesis, but cannot produce more dynamic movements such as jumping on/off from a box. Thus we do not conduct any terrain fitting and adaption as done in [Holden et al. 2017] although it can be easily done once the data is obtained. One possibility to augment the data to synthesize results is to apply physically-based animation to synthesize training data where the characters make use of the preservation of momentum and bouncing effects and augment the data.

11 FUTURE WORK

In addition to the combination of our method with physically-based animation as mentioned above, there are several interesting directions for further research.

One direction can be motion retargeting to quadrupeds of different sizes and morphology. The motion capture of quadrupeds is not straightforward and thus the amount of data is usually small and the motion for the desired body size may not be always available. Wampler et al. [2014] propose an optimization approach for synthesizing locomotion of quadrupeds with different sizes and skeletal structure but the movements are limited to planer frontal movements, and retargeting each motion will be an offline process. It will be interesting to see if neural domain transfer type of approaches based on factorization [Bertinetto et al. 2016] or Resnets [Rebuffi et al. 2017] can be applied such that a wide variation of gaits and movements can be created from a rich set of locomotion of one animal and a small set of exemplar motion of another animal.

Another interesting direction to look into is to compute the adversarial loss of the generated motion and use such loss for the optimization to avoid the ambiguity problem. This has been explored in the physically-based environment [Merel et al. 2017b] and preliminary results for motion capture data [Barsoum et al. 2017], but a framework for interactive character control is yet to be done.

Automatically controlling non-player characters to interact with the user controlled characters or with dynamic obstacles in a complex environment can be an interesting direction of research. One possibility is to apply reinforcement learning for deciding the control signals of our framework.

MANN proved to be powerful for capturing the multi-modal nature of quadruped motion. We would like to explore if the MANN architecture is generally effective on other machine learning tasks with highly multi-modal data.

ACKNOWLEDGMENTS

We thank Yang Liu for early fruitful discussions about the network architecture, and the anonymous reviewers for the constructive comments. We also thank NVIDIA for providing us a DGX-1 for computation. The dog motion data is provided by Bandai-Namco Studios Inc. The quadruped character model was created by Dennis Haupt. This project is partly supported by Google VR Research Faculty Award.

REFERENCES

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A System for Large-scale Machine Learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI'16)*. USENIX Association, Berkeley, CA, USA, 265–283. http://dl.acm.org/citation.cfm?id=3026877.3026899

Mazen Al Borno, Martin De Lasa, and Aaron Hertzmann. 2013. Trajectory optimization for full-body movements with complex contacts. *IEEE transactions on visualization and computer graphics* 19, 8 (2013), 1405–1414.

Okan Arikan and David A Forsyth. 2002. Interactive motion generation from examples. *ACM Trans on Graph* 21, 3 (2002), 483–490.

Emad Barsoum, John Kender, and Zicheng Liu. 2017. HP-GAN: Probabilistic 3D human motion prediction via GAN. *CoRR* abs/1711.09561 (2017). arXiv:1711.09561 http://arxiv.org/abs/1711.09561

Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. 2015. Conditional computation in neural networks for faster models. *arXiv preprint arXiv:1511.06297* (2015).

Luca Bertinetto, João F Henriques, Jack Valmadre, Philip Torr, and Andrea Vedaldi. 2016. Learning feed-forward one-shot learners. In *Advances in Neural Information Processing Systems*. 523–531.

Jinxiang Chai and Jessica K Hodgins. 2005. Performance Animation from Low-dimensional Control Signals. *ACM Trans on Graph* 24, 3 (2005), 686–696.

Xiaobin Chang, Timothy M Hospedales, and Tao Xiang. 2018. Multi-level factorisation net for person re-identification. *arXiv preprint arXiv:1803.09132* (2018).

Simon Clavet. 2016. Motion Matching and The Road to Next-Gen Animation. In *Proc. of GDC 2016*.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. 2015. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). *CoRR* abs/1511.07289 (2015). http://arxiv.org/abs/1511.07289

Stelian Coros, Andrej Karpathy, Ben Jones, Lionel Reveret, and Michiel Van De Panne. 2011. Locomotion skills for simulated quadrupeds. In *ACM Transactions on Graphics (TOG)*, Vol. 30. ACM, 59.

David Eigen, Marc'Aurelio Ranzato, and Ilya Sutskever. 2013. Learning factored representations in a deep mixture of experts. *arXiv preprint arXiv:1312.4314* (2013).

Katerina Fragkiadaki, Sergey Levine, Panna Felsen, and Jitendra Malik. 2015. Recurrent network models for human dynamics. In *Proc. ICCV*. 4346–4354.

Keith Grochow, Steven L Martin, Aaron Hertzmann, and Zoran Popović. 2004. Style-based inverse kinematics. *ACM Trans on Graph* 23, 3 (2004), 522–531.

Chris Hecker, Bernd Raabe, Ryan W Enslow, John DeWeese, Jordan Maynard, and Kees van Prooijen. 2008. Real-time motion retargeting to highly varied user-created morphologies. *ACM Transactions on Graphics (TOG)* 27, 3 (2008), 27.

Daniel Holden, Taku Komura, and Jun Saito. 2017. Phase-functioned neural networks for character control. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 42.

Daniel Holden, Jun Saito, and Taku Komura. 2016. A deep learning framework for character motion synthesis and editing. *ACM Trans on Graph* 35, 4 (2016).

Daniel Holden, Jun Saito, Taku Komura, and Thomas Joyce. 2015. Learning Motion Manifolds with Convolutional Autoencoders. In *SIGGRAPH Asia 2015 Technical Briefs*. Article 18, 4 pages.

Eugene Hsu, Kari Pulli, and Jovan Popovic. 2005. Style Translation for Human Motion. *ACM Trans on Graph* 24, 3 (2005), 1082–1089.

Ting-Chieh Huang, Yi-Jheng Huang, and Wen-Chieh Lin. 2013. Real-time horse gait synthesis. *Computer Animation and Virtual Worlds* 24, 2 (2013), 87–95.

Leslie Ikemoto, Okan Arikan, and David Forsyth. 2009. Generalizing motion edits with gaussian processes. *ACM Transactions on Graphics (TOG)* 28, 1 (2009), 1.

Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. 1991. Adaptive mixtures of local experts. *Neural computation* 3, 1 (1991), 79–87.

Michael I Jordan and Robert A Jacobs. 1994. Hierarchical mixtures of experts and the EM algorithm. *Neural computation* 6, 2 (1994), 181–214.

Ahmad Abdul Karim, Thibaut Gaudin, Alexandre Meyer, Axel Buendia, and Saida Bouakaz. 2013. Procedural locomotion of multilegged characters in dynamic environments. *Computer Animation and Virtual Worlds* 24, 1 (2013), 3–15.

Lucas Kovar and Michael Gleicher. 2004. Automated Extraction and Parameterization of Motions in Large Data Sets. *ACM Trans on Graph* 23, 3 (2004), 559–568.

Lucas Kovar, Michael Gleicher, and Frédéric Pighin. 2002. Motion graphs. *ACM Trans on Graph* 21, 3 (2002), 473–482.

Paul G Kry, Lionel Revéret, François Faure, and M-P Cani. 2009. Modal locomotion: Animating virtual characters with natural vibrations. In *Computer Graphics Forum*, Vol. 28. Wiley Online Library, 289–298.

Manfred Lau and James J Kuffner. 2005. Behavior planning for character animation. In *Proc. SCA*. 271–280.

Jehee Lee, Jinxiang Chai, Paul SA Reitsma, Jessica K Hodgins, and Nancy S Pollard. 2002. Interactive control of avatars animated with human motion data. *ACM Trans on Graph* 21, 3 (2002), 491–500.

Yongjoon Lee, Kevin Wampler, Gilbert Bernstein, Jovan Popović, and Zoran Popović. 2010. Motion fields for interactive character locomotion. *ACM Trans on Graph* 29, 6 (2010), 138.

Sergey Levine and Jovan Popović. 2012. Physically Plausible Simulation for Character Animation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA '12)*. Eurographics Association, Goslar Germany, Germany, 221–230. http://dl.acm.org/citation.cfm?id=2422356.2422388

Sergey Levine, Jack M Wang, Alexis Haraux, Zoran Popović, and Vladlen Koltun. 2012. Continuous character control with low-dimensional embeddings. *ACM Trans on Graph* 31, 4 (2012), 28.

Zimo Li, Yi Zhou, Shuangjiu Xiao, Chong He, and Hao Li. 2017. Auto-Conditioned LSTM Network for Extended Complex Human Motion Synthesis. *arXiv preprint arXiv:1707.05363* (2017).

C. Karen Liu, Aaron Hertzmann, and Zoran Popović;. 2005. Learning physics-based motion style with nonlinear inverse optimization. *ACM Trans. Graph.* 24, 3 (2005), 1071–1081. DOI: https://doi.org/10.1145/1073204.1073314

C. Karen Liu and Zoran Popović'. 2002. Synthesis of complex dynamic character motion from simple anima tions. *ACM Transactions on Graphics* 21, 3 (2002), 408–416.

Libin Liu and Jessica Hodgins. 2017. Learning to schedule control fragments for physics-based characters using deep q-learning. *ACM Transactions on Graphics (TOG)* 36, 3 (2017), 29.

Ilya Loshchilov and Frank Hutter. 2017. Fixing Weight Decay Regularization in Adam. *CoRR* abs/1711.05101 (2017). arXiv:1711.05101 http://arxiv.org/abs/1711.05101

Josh Merel, Yuval Tassa, Sriram Srinivasan, Jay Lemmon, Ziyu Wang, Greg Wayne, and Nicolas Heess. 2017a. Learning human behaviors from motion capture by adversarial imitation. *arXiv preprint arXiv:1707.02201* (2017).

Josh Merel, Yuval Tassa, Dhruva TB, Sriram Srinivasan, Jay Lemmon, Ziyu Wang, Greg Wayne, and Nicolas Heess. 2017b. Learning human behaviors from motion capture by adversarial imitation. *CoRR* abs/1707.02201 (2017). arXiv:1707.02201 http://arxiv.org/abs/1707.02201

Jianyuan Min and Jinxiang Chai. 2012. Motion graphs++: a compact generative model for semantic motion analysis and synthesis. *ACM Trans on Graph* 31, 6 (2012), 153.

Tomohiko Mukai and Shigeru Kuriyama. 2005. Geostatistical motion interpolation. *ACM Trans on Graph* 24, 3 (2005), 1062–1070. DOI: https://doi.org/10.1145/1073204.1073313

Xue Bin Peng, Glen Berseth, and Michiel van de Panne. 2015. Dynamic Terrain Traversal Skills Using Reinforcement Learning. *ACM Trans. Graph.* 34, 4, Article 80 (July 2015), 11 pages. DOI: https://doi.org/10.1145/2766910

Xue Bin Peng, Glen Berseth, and Michiel van de Panne. 2016. Terrain-Adaptive Locomotion Skills Using Deep Reinforcement Learning. *ACM Transactions on Graphics (Proc. SIGGRAPH 2016)* 35, 4 (2016).

Marc H. Raibert and Jessica K. Hodgins. 1991. Animation of dynamic legged locomotion. In *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1991, Providence, RI, USA, April 27-30, 1991*. 349–358. DOI: https://doi.org/10.1145/122718.122755

Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. 2017. Learning multiple visual domains with residual adapters. *arXiv preprint arXiv:1705.08045* (2017).

Charles Rose, Michael F. Cohen, and Bobby Bodenheimer. 1998. Verbs and Adverbs: Multidimensional Motion Interpolation. *IEEE Comput. Graph. Appl.* 18, 5 (1998), 32–40. DOI: https://doi.org/10.1109/38.708559

Alla Safonova and Jessica K Hodgins. 2007. Construction and optimal search of interpolated motion graphs. *ACM Trans on Graph* 26, 3 (2007), 106.

Alla Safonova, Jessica K Hodgins, and Nancy S Pollard. 2004. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. *ACM Trans on Graph* 23, 3 (2004), 514–521.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc V. Le, Geoffrey E. Hinton, and Jeff Dean. 2017. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. *CoRR* abs/1701.06538 (2017). arXiv:1701.06538 http://arxiv.org/abs/1701.06538

Jochen Tautges, Arno Zinke, Björn Krüger, Jan Baumann, Andreas Weber, Thomas Helten, Meinard Müller, Hans-Peter Seidel, and Bernd Eberhardt. 2011. Motion reconstruction using sparse accelerometer data. *ACM Trans on Graph* 30, 3 (2011), 18.

Graham W Taylor and Geoffrey E Hinton. 2009. Factored conditional restricted Boltzmann machines for modeling motion style. In *Proc. ICML*. ACM, 1025–1032.

Graham W Taylor, Geoffrey E Hinton, and Sam T Roweis. 2011. Two distributed-state models for generating high-dimensional time series. *The Journal of Machine Learning Research* 12 (2011), 1025–1068.

Michiel van de Panne. 1996. Parameterized gait synthesis. *IEEE Computer Graphics and Applications* 16, 2 (1996), 40–49.

Kevin Wampler and Zoran Popović. 2009. Optimal gait and form for animal locomotion. In *ACM Transactions on Graphics (TOG)*, Vol. 28. ACM, 60.

Kevin Wampler, Zoran Popović, and Jovan Popović. 2014. Generalizing locomotion style to new animals with inverse optimal regression. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 49.

J.M. Wang, D.J. Fleet, and A. Hertzmann. 2008. Gaussian Process Dynamical Models for Human Motion. *Pattern Analysis and Machine Intelligence, IEEE Trans. on* 30, 2 (Feb 2008), 283–298. DOI: https://doi.org/10.1109/TPAMI.2007.1167

Shihong Xia, Congyi Wang, Jinxiang Chai, and Jessica Hodgins. 2015. Realtime Style Transfer for Unlabeled Heterogeneous Human Motion. *ACM Trans on Graph* 34, 4 (2015), 119:1–119:10.

Yuting Ye and C Karen Liu. 2012. Synthesis of detailed hand manipulations using contact sampling. *ACM Transactions on Graphics (TOG)* 31, 4 (2012), 41.

KangKang Yin, Kevin Loken, and Michiel Van de Panne. 2007. Simbicon: Simple biped locomotion control. In *ACM Transactions on Graphics (TOG)*, Vol. 26. ACM, 105.

Seniha Esen Yuksel, Joseph N Wilson, and Paul D Gader. 2012. Twenty years of mixture of experts. *IEEE transactions on neural networks and learning systems* 23, 8 (2012), 1177–1193.