MotorNet: a Python toolbox for controlling differentiable biomechanical effectors with artificial neural networks

Olivier Codol[1,2,5], Jonathan A. Michaels[1,3,4], Mehrdad Kashefi[1,3,4], J. Andrew Pruszynski[1,2,3,4], Paul L. Gribble[1,2,3]

[1] Western Institute for Neuroscience, University of Western Ontario, Ontario, Canada
[2] Department of Psychology, University of Western Ontario, Ontario, Canada
[3] Department of Physiology & Pharmacology, Schulich School of Medicine & Dentistry, University of Western Ontario, Ontario, Canada
[4] Robarts Research Institute, University of Western Ontario, Ontario, Canada
[5] Correspondence should be addressed to Olivier Codol at codol.olivier@gmail.com.

Abstract

Artificial neural networks (ANNs) are a powerful class of computational models for unravelling neural mechanisms of brain function. However, for neural control of movement, they currently must be integrated with software simulating biomechanical effectors, leading to limiting impracticalities: (1) researchers must rely on two different platforms and (2) biomechanical effectors are not generally differentiable, constraining researchers to reinforcement learning algorithms despite the existence and potential biological relevance of faster training methods. To address these limitations, we developed MotorNet, an open-source Python toolbox for creating arbitrarily complex, differentiable, and biomechanically realistic effectors that can be trained on user-defined motor tasks using ANNs. MotorNet is designed to meet several goals: ease of installation, ease of use, a high-level user-friendly API, and a modular architecture to allow for flexibility in model building. MotorNet requires no dependencies outside Python, making it easy to get started with. For instance, it allows training ANNs on typically used motor control models such as a two joint, six muscle, planar arm within minutes on a typical desktop computer. MotorNet is built on PyTorch and therefore can implement any network architecture that is possible using the PyTorch framework. Consequently, it will immediately benefit from advances in artificial intelligence through PyTorch updates. Finally, it is open source, enabling users to create and share their own improvements, such as new effector and network architectures or custom task designs. MotorNet's focus on higher order model and task design will alleviate overhead cost to initiate computational projects for new researchers by providing a standalone, ready-to-go framework, and speed up efforts of established computational teams by enabling a focus on concepts and ideas over implementation.

1. Introduction

Research on the neural control of movement has a long and fruitful history of complementing empirical studies with theoretical work (Lindsay, 2021). Consequently, a

wide variety of computational model classes have been proposed to explain empirical observations, such as equilibrium point control (Feldman & Levin, 1995; Flanagan et al., 1993; Gribble & Ostry, 2000; Won & Hogan, 1995), optimal control (Shadmehr & Krakauer, 2008; Todorov, 2004), and parallel distributed processing models (Fetz, 1993; Gomi & Kawato, 1993; Jordan & Rumelhart, 1992; Lillicrap & Scott, 2013), commonly known as artificial neural networks (ANNs). Although ANNs were formalized many decades ago, they gained in popularity only recently following their rise to prominence in machine learning (ML; LeCun et al., 2015), as their greater explanatory power and biological realism provide significant advantages against alternative model classes (Gershman & Ölveczky, 2020; Lillicrap et al., 2019; Richards et al., 2019; Saxe et al., 2021).

For neural control of movement, production of theoretical work using ANN models may be viewed as a two-step effort: (1) building a realistic simulation environment that mimics the behaviour of bodily effectors and (2) implement the policy ANNs themselves to train on the environment. Many open-source platforms achieve each of these steps individually, such as MuJoCo (Todorov et al., 2012) or OpenSim (Delp et al., 2007; Seth et al., 2018) for building environments, and JAX, PyTorch or TensorFlow for building and training policy ANNs. However, approaches using these platforms lead to two important impracticalities.

First, the user must rely on two different software platforms, one for the environment and one for the policy ANN. Communication between platforms is not built-in, requiring users to produce custom code to link the policy ANN software with the software implementing the simulation of the environment. This forces significant overhead cost to initiate computational projects and creates barriers to research teams who lack the technical background to build those custom pipelines. A current remedy to this issue is *gymnasium* (Towers et al., 2023), a Python toolbox that provides an interface between policies and environments.

However, *gymnasium* constrains the user to reinforcement learning algorithms (Fujimoto et al., 2018; Lillicrap et al., 2019; Mnih et al., 2015) despite the existence and potential biological relevance of faster training methods such as backpropagation (Lillicrap et al., 2020; Whittington & Bogacz, 2017). The inability to use backpropagation to train policies represents the second impracticality. To date, this has been circumvented by training separate ANNs such as multi-layer perceptrons or recurrent neural networks (RNNs) as "forward models" approximating the behaviour of effectors that are normally implemented in a separate software package (e.g., Lillicrap & Scott, 2013; Willett et al., 2021). This approach does not address the need for custom pipelines, and remains a slow, cumbersome process when iterating over many different policies and environments, because new approximator ANNs must be trained each time.

Solving the issues described above requires both the policy and environment to rely on the same software (no-dependency requirement), and for the environment to allow for backpropagation through itself (differentiability requirement) so that typical gradient-based algorithms may be employed. Ideally, the solution would also be open source, modular for

flexibility of coding and focus on ideas, and have reasonable training speeds on commercially available computers.

We developed MotorNet with these principles in mind. MotorNet is a freely available open-source Python toolbox (https://motornet.org) that allows for the training of ANNs to control arbitrarily complex and biomechanically realistic effectors to perform user-defined motor tasks. The toolbox requires no dependency besides standard Python toolboxes available on *pip* or Anaconda libraries. This greatly facilitates its use on remote computing servers as no third-party software needs to be installed. The environments are fully differentiable, enabling fast and efficient training of ANNs using standard gradient-based methods. It is designed with ease of installation and ease of use in mind, with a high-level and user-friendly application programming interface (API). Its programming architecture is modular to allow for flexibility in model building and task design. Finally, MotorNet is built on PyTorch, which makes innovation in machine learning readily available for use by MotorNet as they are implemented and released by the PyTorch Foundation. Here, we focus on illustrating the scientific use and relevance of the toolbox (the *why*), rather than the underlying API through coding snippets (the *how*), as the latter is more efficiently showcased via interactive, easily updatable online tutorials. The interested reader may consult the full API documentation, including interactive tutorials on the toolbox website at https://motornet.org.

2. Results

2.1. Training an ANN to perform a centre-out reaching task against a curl field.

A canonical experimental paradigm in the study of neural control of movement is the centre-out reaching task with a "curl field" applied at the arm's end point by a robot arm (Conditt et al., 1997; Shadmehr & Mussa-Ivaldi, 1994). In this paradigm, visual targets are placed around a central starting position in a horizontal plane. Participants must move the handle of a robot arm from the starting position to the target that appears on a given trial. During the reaching movement, the robot applies forces at the handle that scale linearly with the velocity of the hand and push in a lateral direction. This leads the central nervous system to adapt by modifying neural control signals to muscles to apply opposite forces to counteract and nullify the lateral forces produced by the robot. Finally, removal of the curl field leaves an opposite after-effect (Shadmehr & Mussa-Ivaldi, 1994). This paradigm is well suited to assess the functionality of MotorNet because it is well understood and extensively documented, and highlights physical, biomechanical, and control properties of human behaviour.

We specified a one-layer RNN composed of 50 gated recurrent units (GRUs; Cho et al., 2014) to control a two degrees of freedom, six muscle planar arm model (*arm26*; Figure 1a; Kistemaker et al., 2006, 2010). The muscles were rigid-tendon, Hill-type muscle models, with "shoulder" mono-articular flexors/extensors, "elbow" mono-articular flexors/extensors, and a bi-articular pair of muscles producing flexion or extension at both joints (see Methods section 4.1.1 and 4.2.1.).

Training the model above took about 13 minutes on a 2022 Mac Studio with an M1 Max central processing unit (Apple Inc., Cupertino CA, USA). Because the *arm26* effector and the centre-out reaching task are particularly common in the motor control literature, they are included in the toolbox as pre-built objects. Consequently, one can recreate the effector instance and the corresponding environment in one line of code for each. Note however that users can easily declare their own custom-made effector and environment objects if desired by subclassing the base *Effector* and *Environment* class, respectively (see below for more details on base classes and subclassing).

Including the implementation of the policy RNN and training routine, the above example can be reproduced with a few lines of code (see tutorial notebooks online), illustrating the ease of use of MotorNet's API. Once the model is trained, it can produce validation results via a forward pass (Figure 1b-c), which can then be saved and analysed afterwards. The results the model produces include joint and cartesian states (positions, velocities), muscle states (lengths, velocities, activations, forces), musculo-tendon states (lengths, velocities), efferent actions (*i.e.*, time-varying muscle drive) and afferent feedback responses (proprioceptive, visual), as well as any activity states from the network if applicable (Figure 1c). Note that actions are different from muscle activations, in that they are input signals to the Ordinary Differential Equation that produces muscle activation (see Methods; Millard et al., 2013; Thelen, 2003).

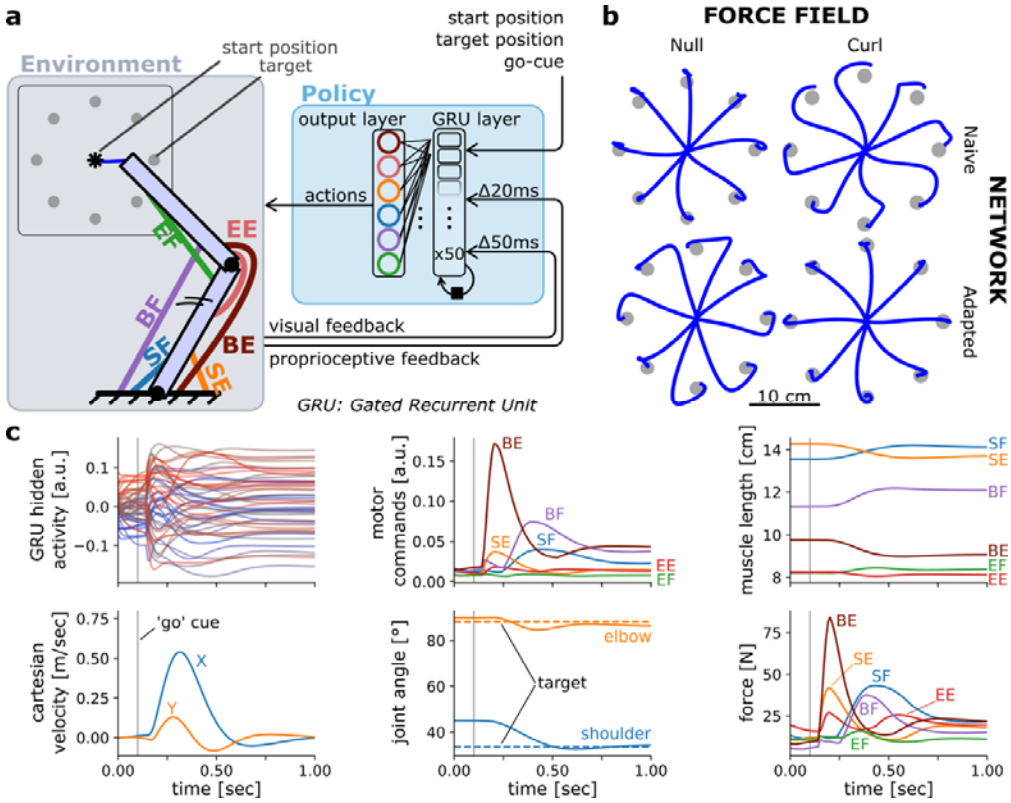

Figure 1: Controlling an arm-like effector in a centre-out reaching task with a curl field. (a) *Schematic of the environment (containing the effector) and policy.* ***(b)*** *Endpoint trajectories of centre-out reaching movements in a null and curl field, for a policy RNN that is untrained (naive) and then*

trained to reach in that curl field. The effector was as defined in Kistemaker et al. (2010) ***(c)*** *Different variables over time during a rightward reaching movement.*

2.2. Structure of MotorNet

Functionally, a MotorNet model can be viewed as a differentiable environment that can directly employ outputs from a policy ANN as action signals. The environment contains an effector, which is actuated according to the action input and in turn outputs information that may be fed back to the ANN (Figure 2). This closed-loop cycle repeats for each timestep. By default, "visual" feedback consists of a vector indicating endpoint cartesian coordinates, while "proprioceptive" feedback consists of a -elements vector of muscle length and velocity, with the number of muscles of the effector. Noise may be added in various parts of the model, such as on descending action signals or on feedback signals. Finally, time delays may be added to feedback signals before they reach the policy ANN. Importantly, the policy ANN may be any PyTorch network, and the MotorNet environments are designed to match standard *gymnasium* API conventions. That is, it is not necessary to create a policy by sub-classing a MotorNet *Policy* object.

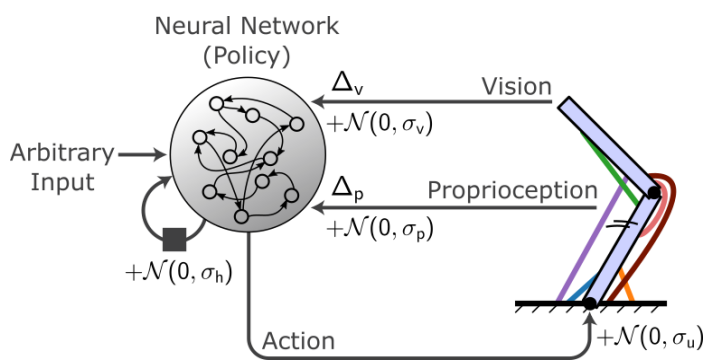

Figure 2: Conceptual organization of a MotorNet model. *A policy ANN receives arbitrary input as well as recurrent connections from itself and sends action signals to an effector embedded in an environment, which in turn sends sensory feedback information. Typically, this feedback will be visual and proprioceptive, and can contain feedback-specific time delays Δ_p and Δ_v. Gaussian noise can be added to the recurrent connection, action signal, and proprioceptive and visual feedback, with specific standard deviation σ_h, σ_u, σ_p, and σ_v.*

2.2.1. Running flow

At runtime, a more detailed representation of the information flow best describes how a MotorNet model behaves (Figure 3a). Models are based on five object classes: *Skeleton*, *Muscle*, *Effector*, *Environment*, and *Policy* objects (Table 1). Each object has its own base class, from which the user can create a custom subclass if desired. MotorNet comes with a set of pre-built subclasses for each, which implement commonly used computational model formalizations (table 1).

	Subclass	Description
Skeleton	*PointMass*	A skeleton with one bone of null length evolving in a plane.
	TwoJointArm	A planar, two-segments skeleton with one hinge joint between the segments and the remaining end of one segment anchored to the world space.
Muscle	*ReluMuscle*	An actuator that produces forces according to a linear piece-wise function of activation. The lower bound of force production is 0.
	RigidTendonHillMuscle	A Hill-type muscle according to the formalization in Kistemaker et al. (2010), adjusted for rigid-tendon dynamics.
	RidigTendonHillMuscleThelen	A Hill-type muscle according to the formalization in Thelen (2003), adjusted for rigid-tendon dynamics.
	CompliantTendonHillMuscle	A Hill-type muscle according to the formalization in Kistemaker et al. (2010).
Effector	*ReluPointMass24*	A planar (2D) *PointMass* with 4 *ReluMuscle* actuators.
	RigidTendonArm26	A *TwoJointArm* with 6 *RigidTendonHillMuscle* actuators.
	CompliantTendonArm26	A *TwoJointArm* with 6 *CompliantTendonArm26* actuators.
Environment	*CentreOutReach*	A centre-out reaching task.
	DelayedReach	A reaching task where movement initiation is signified by the appearance of a "go" cue.
	DelayedMultiReach	A reaching task where movement initiation is signified by the appearance of a "go" cue, and several targets appear in sequence for each trial.
Policy	*PolicyGRU*	An RNN network comprising a user-defined number of layers containing a user-defined number of GRUs.

***Table 1**: Overview of Python base classes and their respective pre-built subclasses in MotorNet. GRU: Gated Recurrent Unit.*

Environment objects are the entry point of the model (Figure 3a). They take arbitrary action inputs, which are then passed on to the *Effector* object. *Effector* objects are essentially wrapper objects that hold the *Muscle* and *Skeleton* objects and handle coordination of information flow between them (Figure 3a-b), as well as concomitant numerical integration to ensure numerical stability. They pass the action signals to the *Muscle* object, which computes forces in return. The *Effector* will adjust those forces using geometry-dependent

moment arms (see section 2.3 for details) and send the resulting generalized forces to the *Skeleton* object. These generalized forces will actualize the *Skeleton*'s joint state, which the *Skeleton* will return to the *Effector* object alongside the equivalent cartesian state. The *Effector* will then return the actuated states to the *Environment* object.

Finally, the *Environment* object will return an observation vector that contains arbitrarily processed information about the states of the *Environment* and *Effector* objects. These can usually be passed on to the policy ANN as input to perform the next forward pass. The *Environment* may maintain a delay buffer, which stores state information for a certain time (according to the Δ_p and Δ_v parameters, Figure 2), allowing the observation vector to be fed time-delayed state information. The *Environment* also outputs an information dictionary, which contains all the instantaneous (*i.e.*, non-delayed) states from the *Environment, Effector, Skeleton* and *Muscle* objects. This allows the user to monitor the true state of the MotorNet model at each timestep.

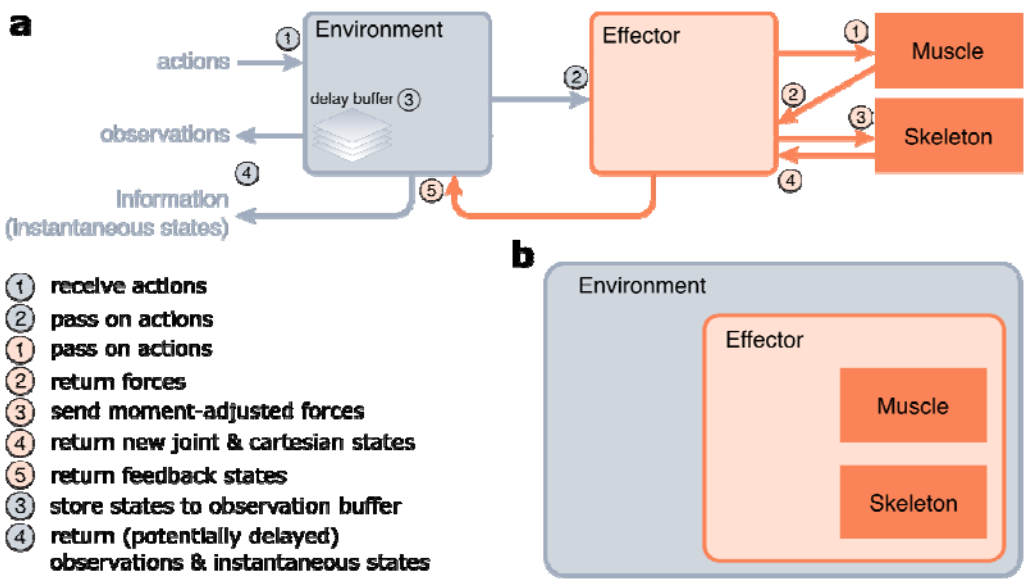

Figure 3: Implementation of MotorNet. (a) *Information flow of a MotorNet model during runtime.* ***(b)*** *Declarative structure of a MotorNet object. Each object instance is held in memory as an attribute of another according to this hierarchical representation, except for the Muscle, and Skeleton instances.*

2.2.2. Object structure

The classes presented above rely on each other to function correctly. Consequently, they must be declared in a sensible order, so that each object instance retains as attribute the object instances on which they rely. This leads to a hierarchical class structure, where each instance lives in the computer memory in a nested fashion with other instances, as laid out in Figure 3b. Note that this does not mean that each class is a subclass of the class that

contains it, but that each contained class is saved as an attribute of the container class. The outermost class is an *Environment* object, which itself is a subclass of *gymnasium*'s *Env* class. The *Environment*, *Effector*, *Skeleton*, and *Muscle* objects are also *torch.nn.Module* subclasses. The *Policy* objects are distinct, in that they do not hold any other MotorNet object as attribute. This independence allows users to create their own neural networks without having to rely on MotorNet's *Policy* object, which allows for more freedom for the user to design any policy that PyTorch can create.

2.3. Biomechanical properties of the effector

The modular structure detailed above allows MotorNet to flexibly compute detailed biomechanical properties of *Effector* objects, such as arbitrary muscle paths (Nijhof & Kouwenhoven, 2000), geometry-dependent moment arms (Murray et al., 1995; Sherman et al., 2013), non-linear muscle activations and passive force production from muscle stretch (Cheng & Scott, 2000; Millard et al., 2013; Thelen, 2003). This enables training ANNs on motor tasks whose dynamics are highly non-linear and close to biological reality. In this section we illustrate some of these biomechanical properties implemented by MotorNet effectors using specific examples. These properties are well-characterised in the biology and are often implemented in realistic biomechanical simulation software.

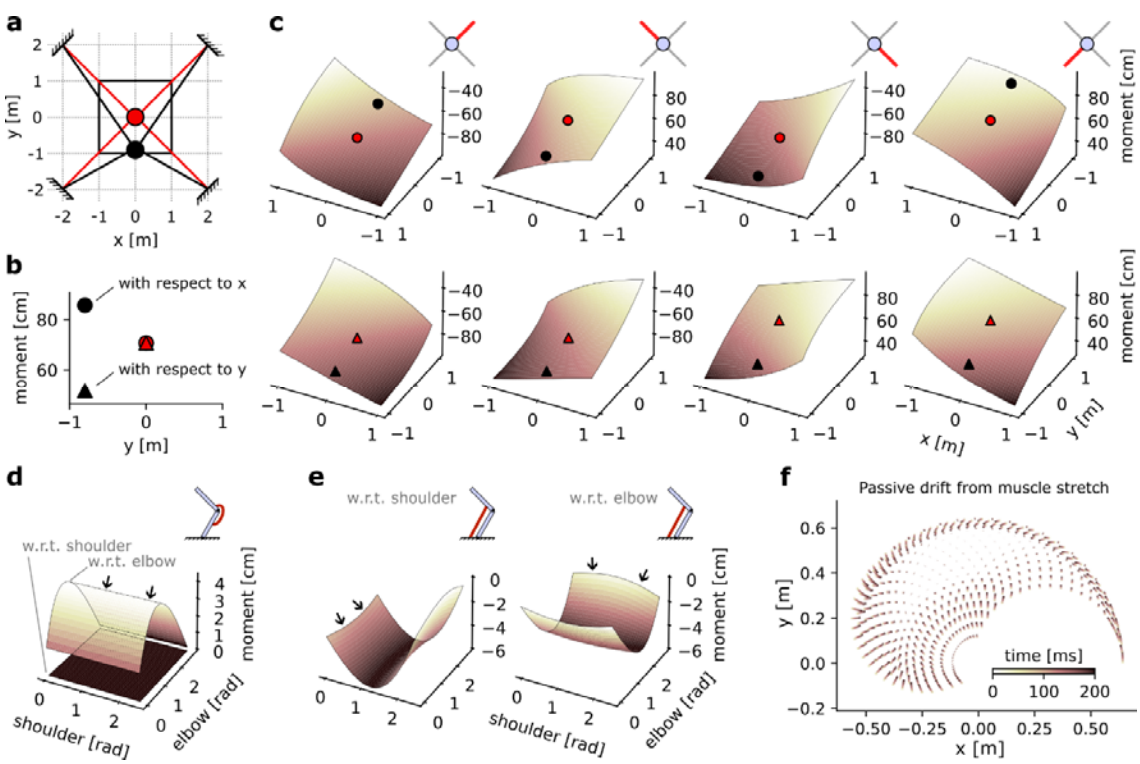

Figure 4: Geometrical properties of an* Effector *object. (a) *Schematic of a point-mass in two positional configurations within a square workspace. The point-mass* Skeleton *was linked to four muscles in a "X" configuration.* ***(b)*** *Moment arm values for the lower left muscle for each of the positional configurations represented in (a), with respect to x and y.* ***(c)*** *Complete moment arm function over the position space for each muscle (columns) and with respect to each DoF. The upper*

and lower row indicate the moment arm with respect to the x and y position, respectively. ***(d)*** *Moment arms of a mono-articular extensor muscle on an arm26.* ***(e)*** *Moment arms of a bi-articular flexor muscle on an arm26.* ***(f)*** *Passive drift in endpoint position of an arm26 similar to Figure 1c due to passive force developed by overstretch Hill-type muscles.*

2.3.1. Assessing moment arms with a simple point-mass effector

The geometrical path – fixation body(s) and fixation point(s) on that body – of each *Muscle* object can be declared by the user, allowing for arbitrary linkage between muscles and bones (see Methods section 4.3, Nijhof & Kouwenhoven, 2000). Using geometric first principles (Sherman et al., 2013), the *Effector* object can then calculate the moment arm of forces produced, which is defined for each muscle as the change in value of the degrees of freedom (DoF) of the skeleton for a given change in the muscle's length (Murray et al., 1995; Sherman et al., 2013). In lay terms, this is the capacity of a muscle to produce a torque on a joint based on the muscle's pulling angle on the bones forming that joint. The relationship between pull angle and torque can intuitively be understood using a door as an example: it is easier to push a door when pushing with an angle orthogonal to that door than in a near-parallel angle to that door.

Moment arms generally vary depending on the positional configuration of the *Effector*. To illustrate this, let us consider a simple case of a point-mass skeleton (one fixation body) with four muscles attached to it in a "X" configuration (Figure 4a). When the point-mass is positioned in the centre of the workspace space (red position in Figure 4a-b), any muscle pulling will change the position of the point-mass equally in the x dimension and in the y dimension. Note that x and y are the DoFs of the point-mass skeleton since they do not have hinge joints. In contrast, if the point-mass is positioned below the central position ($x = 0, y = -0.9$; black position in Figure 4a), a pull from *e.g.*, the lower left muscle will produce a greater change in the x dimension than in the y dimension because of the different muscle alignment (Figure 4b).

The moment arm can then be calculated for all possible positions in the workspace, as represented by the solid black square in Figure 4a. This can be done for each of the four muscles, and each of the two DoFs, resulting in 8 moment arms (Figure 4c). We can see that each moment arm forms a slightly bent hyperplane. Importantly, for each hyperplane the diagonal with constant moment arm lines up with the path formed by the muscle when the point mass is at the centre of the workspace. For instance, the moment arm of the upper right muscle is identical when the point-mass is in position ($x = 1, y = 1$) and in position ($-1, -1$). This is true both with respect to the x DoF (Figure 4c, upper row, leftmost axis) and with respect to the y DoF (Figure 4c, lower row, leftmost axis). Note also that muscles whose shortening leads to an increase in the DoF considered – or inversely whose lengthening leads to a decrease in the DoF – express negative moment arms. For instance, a shortening of the lower right muscle would lead to an increase in the x DoF and a decrease in the y DoF. Or more plainly, a pull from the lower right muscle would bring the point-mass

closer to the lower right corner of the workspace. This leads to the negative moment arm of that muscle with respect to x (Figure 4c, upper row) and positive moment arm with respect to y (lower row).

2.3.2. Moment arms with a two-joint arm

To consider a more complex effector, we assessed the moment arm of two muscles wrapping around a two-joint arm skeleton. We first assessed a mono-articular muscle, that is, a muscle that spans only one joint – here, the elbow. As expected, the moment arm of that muscle with respect to the shoulder joint is always null (black arrows, Figure 4d) regardless of the joint configuration since the muscle does not span that joint. In contrast, the moment arm with respect to the elbow joint varies as the elbow joint angle changes. Finally, as expected from an extensor muscle, the moment arm is positive, indicating that the elbow angle would decrease as the muscle shortens.

In comparison, a bi-articular muscle's moment arm is non-zero with respect to both joints (Figure 4e). This also leads the moment arms with respect to each joint to show a small interaction as the other joint's angle changes, as indicated by a slight "bend" in the hyperplane (black arrows, Figure 4e). Finally, as expected for a bi-articular flexor muscle, the moment arms are negative with respect to both joints, indicating that muscle shortening would result in an increase in joint angle.

2.3.3. Passive drift with Hill-type muscles

Finally, we assessed the positional drift induced by passive forces of Hill-type muscle models (Millard et al., 2013; Thelen, 2003) in an *arm26* effector model. We initialized the model's starting position at fixed intervals across the range of possible joint angles, resulting in a grid of 21-by-21 possible starts. We then simulated the effector with null inputs for 200 ms and plotted the drift in the arm's endpoint position from its original position. Because the model received no input, all forces produced are due to the passive component of the Hill-type muscles, which occurs when the muscle is stretched beyond its slack length (Cheng & Scott, 2000; Millard et al., 2013; Thelen, 2003). We can see that drift is negligible at the centre of the joint space but starts to increase toward the edge (Figure 4f), indicating that the associated joint configurations lead to overstretched muscle lengths and resulting in passive force production. Note that since this phenomenon is dependent on the slack length value of each muscle, which is user-defined, the presence of passive drift is dependent on the user's modelling choices.

2.4. Training ANNs to produce naturalistic behaviour

Now that we can implement biomechanically realistic effectors, we next assessed whether a policy ANN can learn a complex control policy to move those effectors using backpropagation (Jordan & Rumelhart, 1992; Rumelhart et al., 1986). A typical way to ensure the computation learnt by an ANN is functionally meaningful is to test its out-of-

distribution generalization. To assess this, we trained a 1-layer RNN with GRUs controlling an *arm26* model to perform reaching movements in 0.8 sec simulations using the following paradigm. Starting positions and targets were randomly drawn from a uniform distribution across the full joint space. Movements were to be delayed until the occurrence of a visual "go" cue randomly drawn from a uniform distribution spanning the full simulation window. The appearance of the go cue reached the RNN as input after a delay corresponding to the visual feedback delay, which was set at ms (Figure 2; Dimitriou et al., 2013; Pruszynski et al., 2010). In half of trials, no go cue was provided (catch trial), in which case the task effectively reduced to a postural control task. A 100 ms endpoint mechanical perturbation, whose orientation, magnitude, and time were also randomly drawn occurred in half of trials, independently of whether the trial was a catch trial or not. Importantly, the perturbation magnitude was drawn from a uniform distribution ranging between 0 and 4 N. If the perturbation occurred during a catch trial, the distribution ranged between 0 and 8 N. Therefore, the training protocol used for this task largely differed from section 2.1 in that the networks are exposed to a wide range of mechanical perturbations with varying characteristics.

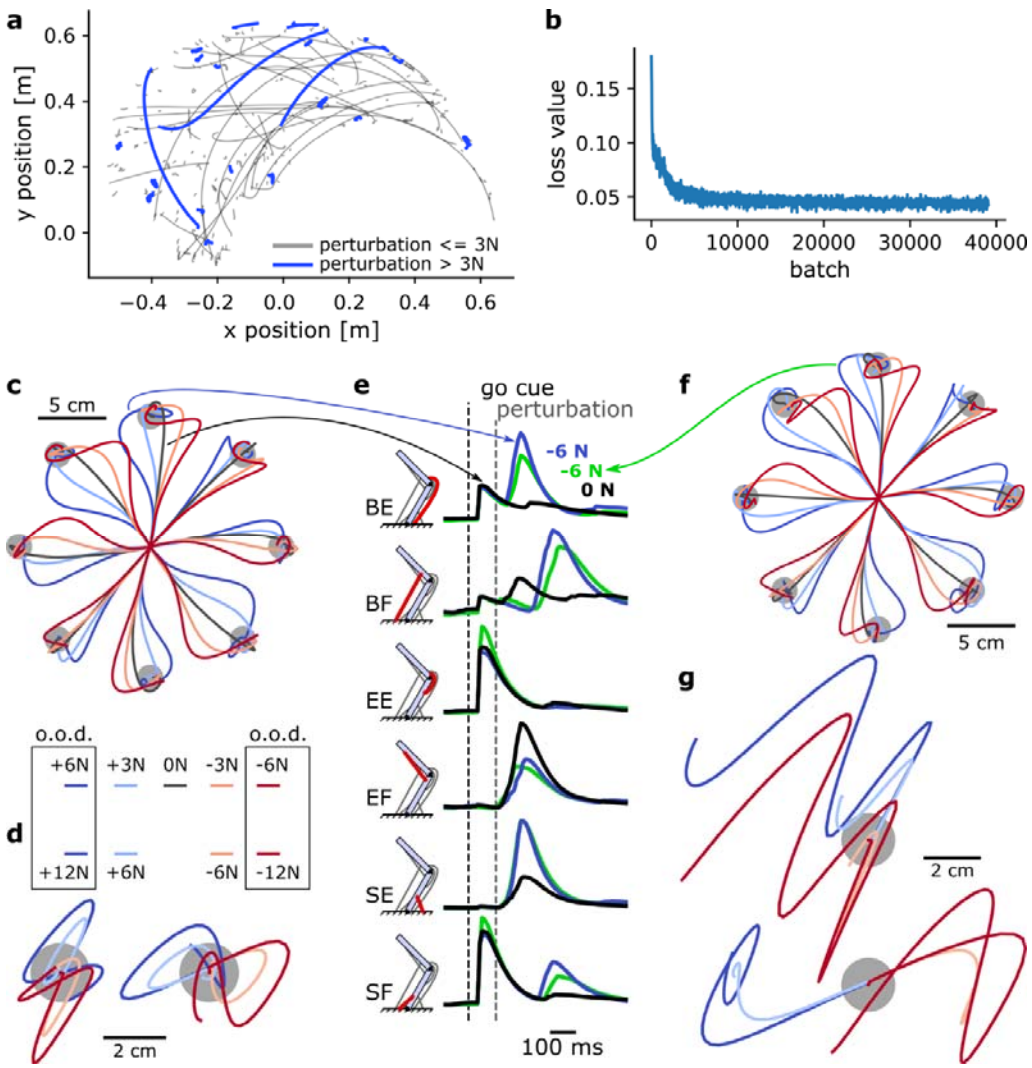

Figure 5: A MotorNet model can learn a control policy that generalizes to out-of-distribution perturbations. (a) *Example endpoint trajectories produced by a network after training.* ***(b)*** *Loss function over training iterations, with a batch size of 1024.* ***(c)*** *Trajectories in a centre-out reaching task with mechanical perturbations applied at the arm's endpoint 120 ms after the "go" cue. The perturbations were orthogonal to the reaching axis passing from the starting position to the target. o.o.d.: out of distribution.* ***(d)*** *Same as (c) for a postural control task. In this task, the network was not provided with a target and therefore only had to remain in the starting position against the perturbations. Mechanical perturbations were in the vertical (left) or horizontal (right) axis.* ***(e)*** *Muscle activation over time for two trajectories in (c) (black and blue lines) and a trajectory in (g) (green line). BE: bi-articular extensor. BF: bi-articular flexor. EE: elbow extensor. EF: elbow flexor. SE: shoulder extensor. SF: shoulder flexor.* ***(f)*** *Reaching task as in (c) for a network never exposed to mechanical perturbations during training.* ***(g)*** *Postural task as in (d) for the same network as in (f). Perturbations were in the vertical (top) or horizontal (bottom) axis.*

The network was trained using the following loss:

$$L = \frac{\sum_{t=1}^{T} L_t}{T} + \lambda \, \|\boldsymbol{W}\|_2$$

$$L_t = \alpha \, L_t^p + \beta \, L_t^m + \gamma \, L_t^h$$

$$L_t^p = \begin{cases} 0, & \|\boldsymbol{x}_t - \boldsymbol{x}_t^*\|_2 < r \\ \|\boldsymbol{x}_t - \boldsymbol{x}_t^*\|_1, & else \end{cases}$$

$$L_t^m = \left(\boldsymbol{u}_t{}^\top \frac{\boldsymbol{f}}{\|\boldsymbol{f}\|_2^2} \right)^2$$

$$L_t^h = \frac{\boldsymbol{h}_t{}^\top \boldsymbol{h}_t}{n} + \kappa \frac{\dot{\boldsymbol{h}}_t{}^\top \dot{\boldsymbol{h}}_t}{n}$$

(eq. 1)

With L the global loss including a kernel regularization term with penalty coefficient $\lambda = 10e^{-6}$, and $\boldsymbol{W}$ the kernel weight matrix of the RNN's hidden layer. The operators $\|\cdot\|_1$ and $\|\cdot\|_2$ indicate the L1 and L2 vector norm, respectively. L_t is the instantaneous loss at time t, with coefficients $\alpha = 2, \beta = 5, \gamma = 0.1$. L_t^p is the positional penalty at time t, with $\boldsymbol{x}_t, \boldsymbol{x}_t^*$ the position and desired position (target) vector, respectively, and $r = 0.01$ the target radius. L_t^m is the muscle activation penalty at time t, with $\boldsymbol{u}_t, \boldsymbol{f}$ two vectors representing muscle activations at time t and maximum isometric force, respectively. Finally, L_t^h is the network hidden activity penalty at time t, with $\boldsymbol{h}_t$ the n-elements vector of GRU hidden activity, $\dot{\boldsymbol{h}}_t$ its time derivative, and $\kappa = 0.05$. While superficially this loss appears complex, a

direct relationship to biology can be drawn for all terms. Essentially, this loss enforces the control policy to be learned using a simple, straightforward rule ("get to the target"), while promoting low metabolic cost from network input connectivity (cost on kernel norm), from the muscles (cost on activation, scaled by muscle strength), and from network activity (cost on hidden activity and its derivative to discourage oscillatory regimes).

Behavioural performance on a training set can be seen in Figure 5a, with trials with a large perturbation ($> 3\ N$) highlighted in blue. This illustrates the rich variability of the training set, encouraging the RNN to learn computationally potent and generalizable solutions to the control problem given the sensorimotor feedback provided (Figure 2). Despite this variability, the loss value decreased smoothly (Figure 5b).

We tested the model's behavioural output in 0.8 sec simulations with a centre-out reaching task. Eight targets were positioned in 45 degrees increments and 10 cm away from a starting position corresponding to a shoulder and elbow angle of 45 and 90 degrees, respectively (Figure 5c). The RNN reached to each of these targets following a visual go cue at 100 ms. 70 ms after the "go" cue was "perceived" (i.e., 70 ms plus the visual feedback delay), a mechanical perturbation was applied at the arm's endpoint and orthogonally to the reaching direction. This perturbation could be either within-distribution (±3 N) or out-of-distribution (±6 N) or null (no perturbation). In all cases, the RNN could correct for the mechanical perturbation, reach to the target, and stabilize (Figure 5c).

Next, we tested the RNN in a postural control task, where it had to bring the arm's endpoint back to the target following a mechanical perturbation (Pruszynski et al., 2014). No go cue was provided. We applied perturbations in either of the four cardinal directions (0°, 90°, 180°, 270°) at 170 ms plus visual delay after the trial started. Again, the set of perturbations for testing outputs included within-distribution magnitudes (±6 N) and out-of-distribution magnitudes (±12 N). In all cases, the RNN could integrate the sensorimotor information to bring the arm's endpoint back into the target (Figure 5d). Interestingly, in some cases this led to an oscillatory trajectory (e.g., for a rightward +12 N perturbation, Figure 5d), indicating that perturbations beyond a given magnitude remain increasingly challenging to control for.

Finally, we compared muscle activations for an upward reach with no perturbation to that of an identical reach with a -6 N perturbation (Figure 5e). We can see that muscle activations are similar before the occurrence of the perturbation, and remain similar immediately after, indicating a time delay in the response. The fastest responses occurred for the bi-articular muscles and the shoulder extensor muscle. Other muscles, particularly the shoulder flexor, showed very delayed or non-existent changes in muscle activation. This illustrates that the RNN's response to a perturbation is not a mere stimulus-driven reactive response, but an integrated response that can delay or withhold the production of counteracting forces if necessary. Note that for the non-perturbed movement (black line in Figure 5e), we can observe the canonical tri-phasic muscle activation pattern reported in empirical studies (Wierzbicka et al., 1986).

To assess how the existence of sensorimotor feedback impacted the control policy acquired by the policy network, we trained a second, identical network to perform the same task but with no mechanical perturbation during training (perturbation-free). Interestingly, following the same amount of training, the model with a perturbation-free network can handle perturbations during reaching relatively well, even up to ±6 N (Figure 5f). We can compare muscle activations for an upward reach with a -6 N perturbation to that of the same movement in the network trained with perturbations (Figure 5f, green versus blue lines). Even though kinematics appeared superficially similar (Figure 5c, f), this comparison shows that muscle activations tend to differ in response to a perturbation (Figure 5e), suggesting that the perturbation-free network might learn a slightly different control policy. Testing the perturbation-free network on the postural task shown in Figure 5d emphasizes this difference (Figure 5g). The perturbation-free network is much less capable of stabilizing against the forces than its perturbation-trained counterpart.

Therefore, even though the mere existence of a sensorimotor feedback input can help handle simple perturbations (Figure 5f), exposing the model to perturbations during training does provide the network with additional information to learn a more robust control policy. Overall, these simulations show that MotorNet can train ANNs to reliably find a control policy for the effector. Importantly, the resulting networks learn generalizable control policies that integrate sensorimotor feedback into its computation. This also illustrates the importance of the training procedure to which the network is exposed to produce these control policies (Driscoll et al., 2022).

2.5. Effector Geometry Defines Preference Distribution of Firing Rates: A Replication Study

Finally, to assess MotorNet's capacity to replicate established results in the literature, we sought to reproduce key observations from Lillicrap and Scott (2013). In that study, the authors show that training an RNN to perform a simple centre-out reaching task using an arm model similar to the arm26 in Figure 1a results in the RNN neurons displaying a preferential movement direction (PMD) where they are more likely to fire. The distribution of PMDs was asymmetrical, with a greater proportion of neurons firing for reaches around 135 degrees and 325 degrees, matching empirical observations from non-human primate electrophysiological recordings in the primary motor cortex (Scott et al., 2001). Next, they showed that this asymmetrical representation of PMDs during reaching movements did not occur when RNNs were trained to control an effector that lacked the geometrical properties of an arm such as illustrated in Figure 4c-e and section 2.1. Specifically, they compared the PMD distribution of RNN neurons controlling a point-mass (no geometry) against that of an arm26 (geometry present).

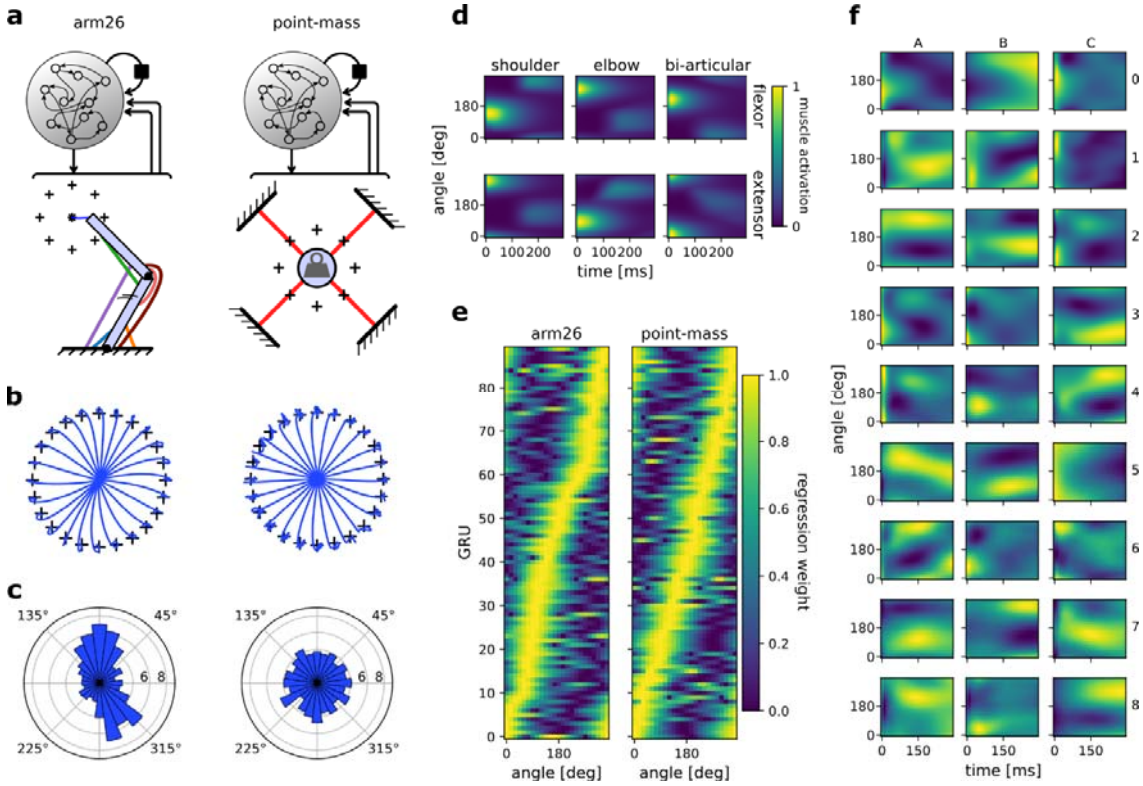

Figure 6: The distribution of preferential movement direction tuning is sensitive to the geometry of the effector. (a) *Schematic of the two models compared. The RNNs and their architecture were identical, but the effector differed, with one RNN controlling a two-joint arm26 (left) and the other controlling a point-mass (right).* ***(b)*** *Centre-out reaching trajectories to 24 targets for the arm26 (left) and point-mass (right) model.* ***(c)*** *Distribution of preferential movement directions (PMDs) for the arm26 (left) and point-mass (right) model. The PMDs were determined by regression of each GRU's hidden activity averaged over time against reach angle (see Methods for details).* ***(d)*** *Normalized muscle activations across reaching angles and for the 300 ms following the "go" cue for the arm26 model.* ***(e)*** *Normalized β coefficients of the regression models used for (c). The GRUs were ordered according to the angle of their maximum β value. Note that the "ridge" of maximum β yields roughly a straight line for the point-mass model, while it yields a crooked line for the arm26, indicative of a representation bias.* ***(f)*** *Hidden activity over time and across reaching angles for a random sample of GRUs in the arm26 model.*

We sought to reproduce the two results outlined above. First, we trained an RNN composed of 90 GRUs in a single layer to control for an arm26 (Figure 6a see Methods section 4.5). Because our RNN employs GRUs instead of a multi-layer perceptron, 90 units were sufficient to efficiently train the network to perform the task, as opposed to up to 1000 perceptron nodes in the original study. We also increased the number of targets from 8 to 24 to obtain a finer resolution over movement direction in our analyses (Figure 6b).

Following training, we first ensured that muscle activation patterns in the arm26 effector were like those reported in the original study (Figure 6d). Regarding network activity, we

observed a great variety of activation patterns over movement direction (Figure 6f). Some GRUs showed a preference for timing (e.g., neuron A4, C5), while others showed a strong preference for reaching direction that was sustained over time (neuron C3, A2). Finally, most neurons showed a mixed preference for encoding time and reaching direction (neuron C8, A8). This heterogeneous set of responses matches empirical observations in non-human primate primary motor cortex recordings (Churchland & Shenoy, 2007; Michaels et al., 2016) and replicate similar visualizations from previously published work (Fortunato et al., 2023; Lillicrap & Scott, 2013; Safaie et al., 2023).

We then assessed each GRU's PMD using linear regression (see methods) and sorted them based on their PMD before plotting the tuning curve of each neuron. The resulting colormap (Figure 6e, left panel) yields a "ridge" of maximal activity whose peak varies across reach angle, forming a crooked line, illustrating a representational bias. This crooked ridge line was not observed in an RNN trained to control for a point-mass effector instead using an identical training procedure and analysis (Figure 6e, right panel). We replicated this procedure with 7 more RNNs for each model, resulting in a total of 8 RNNs trained on an arm26 and 8 RNNs trained on a point-mass. We determined each GRU's PMD and averaged the resulting polar histogram across each RNN (Figure 6c). The same bias was reproduced invariably for the RNNs controlling an arm26 effector (Figure 4d-e, Figure 6a), while it failed to arise for those controlling a point-mass (Figure 4a-c, Figure 6a). Therefore, these results mimic the observations made in the original study (Lillicrap & Scott, 2013), specifically, that RNNs controlling an effector with no arm-like geometrical properties will not result in the biased PMD representation during reaching movements commonly observed in non-human primate electrophysiological studies (Scott et al., 2001; Scott & Kalaska, 1997).

3. Discussion

3.1. Iterating quickly through the model development cycle

In the field of machine learning, an established best practice is to iterate quickly around a cycle of (1) formulating an idea, (2) implementing that idea in functionally efficient code, and (3) testing the idea through running the simulations. The results of the simulations can then be leveraged to adjust the idea, thus closing the loop, and enabling iterative refinement of a model. This [idea → code → test → idea] cycle is reminiscent of the [hypothesis → design task → test → hypothesis] cycle in empirical work, also known as the hypothetico-deductive method. An important practice in ML is to ensure that one iteration of that cycle is quick enough, because producing an efficient model may require many such iterations. Based on this framework, a way to view MotorNet is to improve iteration speed through this cycle. The modular architecture of MotorNet enables users to alter specific aspects of the model while keeping everything else identical. Therefore, user capacity to proceed through the "implementation" step is enhanced.

3.2. Advantages

3.2.1. Expandability

MotorNet naturally allows users to create and tune objects to fit individual requirements. This makes the toolbox easily expandable to add novel models that are not pre-built in the original distribution. This flexibility will likely vary depending on the goal (Figure 7). Some extensions only require adjusting parameter values of existing object classes, such as editing the Arm26 *Skeleton* class to match the arm of a non-human primate. Other extensions will require subclassing, such as creating an *Effector*for an eyeball, which might require special geometric properties building on the point-mass *Skeleton* object (Table 1). Conversely, effectors that stray away from typical vertebrate effectors will likely prove more challenging, such as an octopus arm, because they do not rely on bones. Importantly, while all these extensions vary in the difficulty of their implementation, each has the capacity to fit and work harmoniously within the framework of the MotorNet architecture.

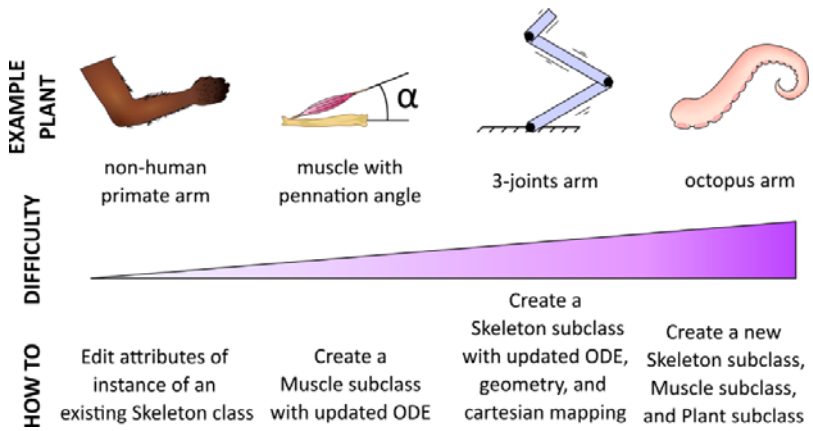

Figure 7: MotorNet is expandable. *MotorNet allows for new features to be implemented through subclassing.*

3.2.2. Open source

Typically, when motor control researchers want to create canonical models, they must implement their own version of said model based on methodological descriptions of previously published scientific articles. However, because MotorNet is open source, individual contributions can easily be shared online for the benefit of others. For instance, if a researcher creates a *Muscle* class with a parametrizable pennation angle (Millard et al., 2013; Thelen, 2003), future researchers and team will not have to re-create their own implementation of the same object anymore. This also allows more dynamical peer-checking, avoiding dissemination of errors and improving consistency of model implementations. In other words, MotorNet will be able to benefit from community driven incremental work through open-source practices.

3.2.3. Innovation scalability

For the past several years, ML has been standing out as one of the most dynamic research fields, achieving breakthroughs and successfully scaling innovative work toward solving

everyday problems. It would be challenging for MotorNet to keep up with the pace of ML innovation to provide users with implementations of the latest architectures and algorithms. Rather, we rely on PyTorch to build policies. This ensures that any innovation in model design quickly finds its way to a viable MotorNet implementation, because PyTorch capabilities allow for fast adaptation aligned with progress in ML. Generally, MotorNet is built with the following logic in mind: anything PyTorch can build, MotorNet should be able to use as a policy.

3.2.4. *gymnasium*-compliant interfacing

The MotorNet *Environment* class is a subclass of *gymnasium*'s *Env* base class and abides by its associated API (Towers et al., 2023). Consequently, MotorNet environments are by design compatible with any Python toolbox that works with *gymnasium*, which is a standard and popular interfacing toolbox to link reinforcement learning agents with environments. It is very well-documented and widely used, which will ensure that users who wish to employ reinforcement learning to control MotorNet environments will be able to do so relatively effortlessly.

3.3. Limitations

3.3.1. Collision physics

Typical biomechanical software distributions implement some form of collision physics in their physics engine (Delp et al., 2007; Seth et al., 2018; Todorov et al., 2012). This is not the case for MotorNet.

3.3.2. Complex biomechanical features

Some biomechanical software distributions such as OpenSim propose a large array of joint types such as hinge joints or rotational joints, and complex muscle paths such as wrap points that trigger only when the muscle collides with them (Delp et al., 2007; Seth et al., 2010, 2018). While these features increase the realism of a biomechanical model, MotorNet does not yet implement these types of features. In practice, this constrains what types of effectors MotorNet can realistically implement and adding some of these features is under consideration.

3.4. Future considerations

As an open-source, freely available Python toolbox, MotorNet is subject to change over time. Some of the limitations outlined above are considered as future routes for improvement. Additionally, we hope that individual contributions will help refine and extend the capabilities of the toolbox as well. In this section we outline prospective improvements for implementation and release in the main distribution.

3.4.1. Spinal Compartment

It is becoming increasingly evident that spinal contribution plays a prominent role in motor control beyond the typically considered spinal reflex (Reschechtko & Pruszynski, 2020; Weiler et al., 2019). One may consider that supraspinal control interacts with spinal contribution to define a motor control policy (Loeb, 2021). Within MotorNet, this suggests that a policy's latent dynamics will be significantly impacted by the presence of a spinal compartment. Consequently, it may be worthwhile to implement one such spinal compartment to explore the consequences of such biological design, especially with regards to upstream computation (Cisek, 2019). Importantly, this compartment may be designed as a module within the controller downstream from the ANN, that instantiates arbitrarily detailed computation according to empirical studies. This could include processing of top-down and/or bottom-up information as a movement unfolds (Reschechtko & Pruszynski, 2020) with appropriate time delays in place. Ultimately, the scientific question at hand will dictate the complexity desiderata for the spinal compartment implementation. Interestingly, this two-modules design within the controller leans toward the more general concept of modular architectures, which can be powerful for understanding multi-region interactions within the central nervous system (Michaels et al., 2020).

3.4.2. Modular policies

A deeply established idea in neuroscience is that distinct regions will perform different computations, and thus that a complex system may not be considered as a uniform, fully connected network (Abbott & Svoboda, 2020; Keeley et al., 2020; Pesaran et al., 2021; Semedo et al., 2020). This is also true for the motor control system, where using a modular network architecture with controlled communication between each module has been shown to have more explanatory power than a non-modular system (Michaels et al., 2020). Therefore, a potential development for MotorNet is to include a model class with a modular architecture to study how cross-regions networks work to enable neural control of the body.

3.4.3. Muscle models

Most published work in motor control relies either on Hill-type muscle models (Bhushan & Shadmehr, 1999; Kistemaker et al., 2006, 2010; Nijhof & Kouwenhoven, 2000) or direct torque actuators (Lillicrap & Scott, 2013) similar to the ReLu muscle that MotorNet provides. However, despite its popularity, even the more-detailed Hill-type muscle remains a phenomenological model of real muscle behaviour, which can easily show its limits when trying to understand how the brain controls movement (Blum et al., 2020). Alternative muscle model formalizations exist, such as the Distribution-Moment muscle model (Zahalak, 1981), which may be worth implementing within MotorNet as well.

4. Methods

4.1. General modelling design

This section describes modelling elements that were used for several models in this study. For all models, the timestep size was 0.01 sec, and a proprioceptive delay of $\Delta_p = 20$ ms and visual delay of $\Delta_v = 50$ ms were used (Figure 2). Effectors were actuated using numerical integration with the Euler method.

4.1.1. Arm26 model

The arm26 model used in this study is available online on the open-source toolbox code under the *RigidTendonArm26 Effector* class. It is briefly described below for convenience.

The skeleton of the arm26 models are according to the formalization proposed in Gomi & Kawato (1997), equations 1, 3, 5-7. Parameter values are as in table 2.

Parameter	Upper arm	Forearm
Mass (kg)	1.82	1.43
Centre of gravity (m)	0.135	0.165
Inertia (kg.m^2)	0.051	0.057
Length (m)	0.309	0.333

***Table 2**: skeleton parameters for the arm26 model, taken from Nijhof & Kouwenhoven (2000).*

The skeleton was actuated by six rigid-tendon versions of Hill-type muscle actuators: a shoulder flexor, a shoulder extensor, an elbow flexor, an elbow extensor, a bi-articular flexor, and a bi-articular extensor. Their parameter values are defined in table 3.

Muscle	Maximum isometric force (N)	Tendon length (m)	Optimal muscle length (m)
Shoulder Flexor	838	0.039	0.134
Shoulder Extensor	1207	0.066	0.140
Elbow Flexor	1422	0.0172	0.092
Elbow Extensor	1549	0.187	0.093
Bi-articular Flexor	414	0.204	0.137
Bi-articular Extensor	603	0.217	0.127

***Table 3**: parameters for the Hill-type muscle actuators used in the arm26, taken from Kistemaker et al. (2010).*

The full formalization of the Hill-type muscles can be found in Thelen (2003) equations 1-7, and with the parameter values used in that study. When different parameters were provided for young and old subjects, the values for young subjects were used (Thelen, 2003, table 1)

While in custom-made *Effector* objects the moment arms of each muscle are computed based on geometric first principles (Figure 4d-f; Sherman et al., 2013), in the *RigidTendonArm26* class the moment arms are approximated as described in Kistemaker et al. (2010), equations A10-A12, with parameters for this study defined in table 4.

Muscle	a_0	a_{1e}	a_{1s}	a_{2e}
Shoulder Flexor	0.151	0	-0.03	0
Shoulder Extensor	0.2322	0	0.03	0
Elbow Flexor	0.2859	-0.014	0	-4.0e-3
Elbow Extensor	0.2355	0.025	0	-2.2e-3
Bi-articular Flexor	0.3329	-0.016	-0.3	-5.7e-3
Bi-articular Extensor	0.2989	0.03	0.03	-3.2e-3

Table 4*: parameters used to compute moment arms in the arm26 models with moment arm approximation, taken from Kistemaker et al. (2010).*

4.1.2. Point-mass model

The point-mass model used in this study is available online on the open-source toolbox code under the *ReluPointMass24 Effector* class. It is briefly described below for convenience.

The point-mass had a mass of $m = 1$ kg. Its actuation followed an ordinary differential equation such that $\ddot{\boldsymbol{x}} = \boldsymbol{f}/m$ with $\ddot{\boldsymbol{x}}, \boldsymbol{f}$ the 2-elements cartesian acceleration vector at time t and the 2-elements force vector applied at time t, respectively.

The forces were produced by four linear muscle actuators, whose formalization is available online on the open-source toolbox code under the *ReluMuscle* muscle class. Each muscle's force production f is a linear piecewise function of its activation a, scaled by its maximum isometric force $f_{max} = 500$ N:

$$f(a) = \begin{cases} 0, & a \leq 0 \\ f_{max} \cdot a, & 0 < a < 1 \\ f_{max}, & a \geq 1 \end{cases}$$

The activation function was the same as for the Hill-type muscles used in the arm26 model, and can be found in Thelen (2003), equations 1-2.

The four muscles were fixed to the point-mass in a "X" configuration (Figures 4a, 6a) with the first fixation point for the upper right, lower right, lower left, and upper left muscle being respectively $(x = 2, y = 2), (2, -2), (-2, -2), (-2, 2)$. The second fixation point of each muscle was on the point-mass, therefore moving in general coordinates alongside the point-mass (Figure 4a).

4.1.3. Policy network architecture

All policy networks used in this study consisted of one layer of GRUs with a sigmoid recurrent activation function and a $tanh$ activation function. Kernel and recurrent weights were initialized using Glorot initialization (Glorot & Bengio, 2010) and orthogonal initialization (Hu et al., 2020), respectively. Biases were initialized at 0.

The GRU layer was fully connected to an output layer of perceptron nodes with a sigmoid activation function. The output layer contains one node per descending action signal, or equivalently one node per muscle. The output layer's kernel weights were initialized using a random normal distribution with a standard deviation of 0.003, and its bias was initialized at

a constant value of -5. Because the output activation function is a sigmoid, this initial bias forces the output of the policy to be close to 0 at the start of initialization, ensuring a stable initialization state.

For all networks used in this study, a 2-elements vector of (x, y) cartesian coordinates for the start position and target position were provided as input, alongside a go-cue, resulting in a 5-element input vector. The go-cue was a "step" signal whose value changed from 1 to 0 when the movement should be initiated.

4.1.4. General training design

During training, the models reached from a starting position drawn from a random uniform distribution across the full joint space to a target position drawn from a random uniform distribution as well. The occurrence time of the go-cue was drawn from a random uniform distribution across the full simulation duration. In 50% of simulations, no go-cue was provided (i.e., a catch trial) to ensure the network learnt to wait for the go-cue and avoided any anticipatory activity. The desired position $\boldsymbol{x}^*$ was set to be the start position until the go cue was provided, at which point $\boldsymbol{x}^*$ was defined as the target position. Note that the go-cue was treated as a visual signal. Therefore, while the desired position $\boldsymbol{x}^*$ was updated immediately as the go-cue was provided (with no time delay), the network was informed of the go-cue occurrence via a change in the target position input and go-cue input only following the visual feedback delay Δ_v. Depending on the models, additional training manipulations were also applied, as described in the sections below.

4.2. Centre-out reaches task against a curl field

4.2.1. Model

The effector type used to learn to reach against a curl field was an arm26 model as described in section 4.1.1. The policy was as described in section 4.1.3., with the GRU layer containing $n = 50$ units.

4.2.2. Training

The model was trained according to the procedure in section 4.1.4. with the loss described in eq. 1, using a kernel regularization $\lambda = 10e^{-6}$, coefficients $\alpha = 2, \beta = 5, \gamma = 0.1, \kappa = 0.05$, and target radius $r = 0.01$ m. The model was trained on 7680 batches with a batch size of 64, on simulations of 1 sec.

The model was trained according to section 4.1.4., except that the go-cue time was fixed at 100 ms from the start of the simulation. Following initial training, the model was then tested as described in the next section to produce the "naïve" behaviour shown in Figure 1b-c. Following testing, training was then resumed, but employing the curl-field, fixed starting position, and set of 8 targets used in testing. 50% simulations were still catch trials, as in the initial training session. This second training session lasted 768 batches with a batch size of 64. Finally, following this second training session, the model was tested again, to produce the "adapted" behaviour of Figure 1b.

4.2.3. Testing

The model was tested in 1 sec simulations against a null field, and against external forces applied at the arm's endpoint that produced a counter-clockwise curl field:

$$\boldsymbol{f}_t = b \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \dot{\boldsymbol{x}}_t \qquad \text{(eq. 2)}$$

With $\dot{\boldsymbol{x}}_t$ the 2-elements cartesian velocity vector at time t, and $b = 8$ a scalar defining the strength of the curl field. In the null field, we have $b = 0$.

The testing procedure consisted of 8 centre-out reaches from a fixed starting position at a shoulder and elbow angle of 45° and 90°, respectively, to 8 target positions 10 cm away and distributed in increments of 45° around the starting position (Figure 1b). This set of simulations were repeated against a null field and against the curl field in eq. 2, resulting in a total of 16 reaches. For all testing simulations, the go-cue time was fixed at 100 ms from the start of the simulation and no catch trials were employed.

4.3. Biomechanical properties of the effector

The point-mass model used was as described in section 4.1.2. The arm26 model used was as described in section 4.1.1., except that the moment arms were not approximated based on the parameters of table 4, but computed based on the geometry of the muscle paths (Nijhof & Kouwenhoven, 2000; Seth et al., 2010; Sherman et al., 2013). Accordingly, the muscle paths were manually declared by defining how many fixation points each muscle has, and on which bone and where on each bone each point fixes.

MotorNet handles declaration of these paths using a relative reference frame for each fixation point (Seth et al., 2010). Specifically, a fixation point on a bone will have two coordinates. The first coordinate defines how far along the bone the point is, from the bone's origin, e.g., the shoulder for the upper arm (Figure 8). The second coordinate defines how far the point deviates from the bone orthogonally. If the fixation point is an anchor point, that is, it is not fixed on a bone but on the world space, then general coordinates (x, y) are used (color-coded in green in Figure 8). These anchor points are important to ensure that the effector can be actuated with respect to the environment. The full set of coordinates defining the model's muscle paths are indicated in table 5 and are derived from Nijhof & Kouwenhoven (2000).

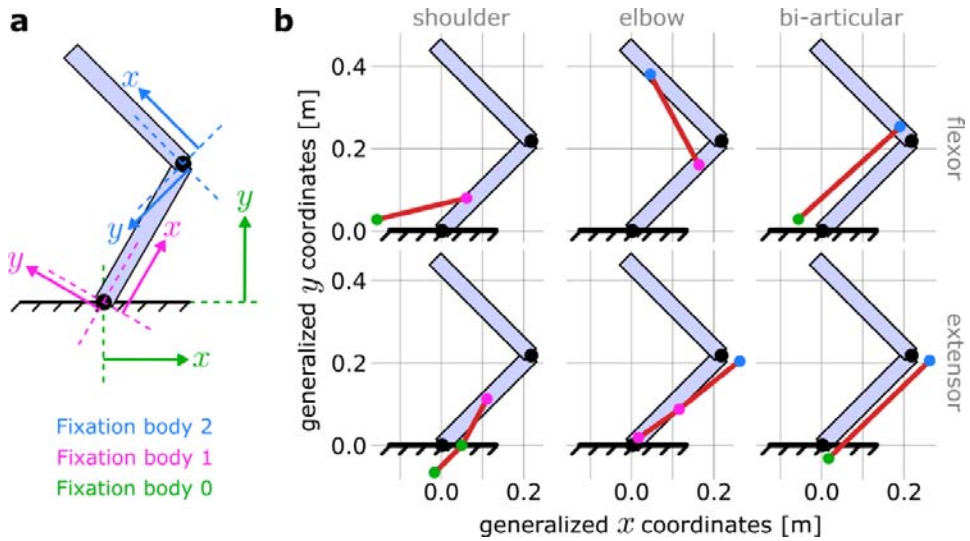

Figure 8: Coordinate frames for declaring muscle paths in MotorNet. (a) *MotorNet handle muscle paths using coordinate frames relative to the bone on which a fixation point is. The world space is indexed as the fixation body "0" and its coordinate frame is the general coordinate system.* ***(b)*** *Schematic illustration of the muscle paths used for the arm26 arm26 model with no moment arm approximation described in section 4.3 and table 5, for a shoulder and elbow angle of 45° and 90°, respectively.*

Muscle	Fixation point	Fixation body	First coordinate (m)	Second coordinate (m)
SF	1	0 (world)	-0.15	0.03
	2	1 (upper arm)	0.094	0.017
SE	1	0 (world)	-0.013	-0.07
	2	0 (world)	0.05	0
	3	1 (upper arm)	0.153	0
EF	1	1 (upper arm)	0.23	0.001
	2	2 (forearm)	0.231	0.01
EE	1	1 (upper arm)	0.03	0
	2	1 (upper arm)	0.138	-0.019
	3	2 (forearm)	-0.04	-0.017
BF	1	0 (world)	-0.052	0.033
	2	2 (forearm)	0.044	0.001
BE	1	0 (world)	0.02	-0.028
	2	2 (forearm)	-0.04	-0.017

Table 5*: Muscle paths for the arm26 model with no moment arm approximation.*

4.4. Training ANNs to produce naturalistic behaviour

4.4.1. Model

The two models used to produce Figure 5 were arm26 models as described in section 4.1.1. For both models, the policy was as described in section 4.1.3., with the GRU layer containing $n = 110$ units. In addition, excitation and GRU hidden activity noise were added, with values $\sigma_u = 10e^{-3}, \sigma_h = 10e^{-4}$, respectively.

4.4.2. Training

The models were trained with the loss described in eq. 1, using a kernel regularization $\lambda = 10e^{-6}$, coefficients $\alpha = 2, \beta = 5, \gamma = 0.1, \kappa = 0.05$, and target radius $r = 0.01$ cm. The model was trained on 27,000 batches of size 1024, on simulations of 800 ms.

In one of the two models, which we refer to as the "perturbation-free" model, the training procedure was as described in section 4.1.4. In the second model, which we refer to as the "perturbation-trained" model, a 100 ms endpoint mechanical perturbation was added to the training procedure. The perturbation occurred in 50% of trials, independently of whether the trial was a catch trial or not, and its orientation and time were randomly drawn as well. The magnitude of the perturbation was drawn from a uniform distribution ranging between 0 and 4 N. If the perturbation occurred during a catch trial, the distribution ranged between 0 and 8 N.

4.4.3. Testing

Both the perturbation-trained and perturbation-free models were tested in 800 ms simulations in two distinct tasks, a centre-out reaching task and a postural task.

In the centre-out reaching task, 8 targets were positioned in 45 degrees increments and 10 cm away from a starting position corresponding to a shoulder and elbow angle of 45° and 90°, respectively (Figure 5c, g). The visual go cue was provided at 100 ms following the simulation start. 70 ms after the go-cue was "perceived" (i.e., 70 ms plus the visual feedback delay Δ_v), a mechanical perturbation was applied at the arm's endpoint and orthogonally to the reaching direction. This perturbation could be either within-distribution (±3 N) or out-of-distribution (±6 N) or null (no perturbation).

In the postural control task, no go cue was provided, and the arm's endpoint was pushed away from the start position by the mechanical perturbation at 170 ms plus visual delay Δ_v after the simulation started. We applied perturbations in either of the four cardinal directions (0°, 90°, 180°, 270°). Again, the set of perturbations for testing outputs included within-distribution magnitudes (±6 N) and out-of-distribution magnitudes (±12 N).

4.5. Effector Geometry Defines Preference Distribution of Firing Rates: A Replication Study

4.5.1. Models

All arm26 and point-mass effectors used to produce Figure 5 were as described in section 4.1.1. and 4.1.2., respectively. For all models, the policy was as described in section 4.1.3., with the GRU layer containing $n = 90$ units.

4.5.2. Training

All models were trained with the loss described in eq. 1, using a kernel regularization $\lambda = 10e^{-6}$, coefficients $\alpha = 2, \beta = 5, \gamma = 0.1, \kappa = 0.05$, and target radius $r = 0$. The models were trained on 38,400 batches of size 64, on simulations of 800 ms. The training procedure was as described in section 4.1.4.

4.5.3. Testing

The testing procedure consisted of 8 centre-out reaches in 800 ms simulations. Simulations started from a fixed position at a shoulder and elbow angle of 45° and 90° for the arm26 models, and at an $(x = 0, y = 0)$ cartesian position for the point-mass models. Reaches were to 24 target positions 10 cm away and distributed in increments of 15° around the starting position (Figure 6b). For all testing simulations, the go-cue time was fixed at 100 ms into the simulation and no catch trials were employed.

4.5.4. Analysis

To obtain the preferential movement direction of each GRU, we averaged each unit's hidden activity in a 150 ms time window starting when the go cue was input to the network (i.e., following visual feedback delay Δ_v) for each reaching direction independently, and regressed that average to a diagonal design matrix encoding the reach direction. The absolute value of the resulting regression coefficients was then normalized between 0 and 1, and neurons were sorted according to these normalized coefficients to produce Figure 6e.

As mentioned in the results section, we trained 8 networks to control an arm26 and 8 networks to control a point-mass. For each network, we took the count of GRUs whose normalized regression coefficient is maximal for each target considered and averaged that count across all 8 networks to produce Figure 6c.

5. Contributions

OC, JAM, and PLG conceptualized the toolbox; OC implemented the toolbox; OC, JAM, MK, JAP, and PLG tested the toolbox and designed the simulation paradigms; OC and PLG performed the simulations; OC, JAM, MK, JAP, and PLG interpreted the results; OC made the figures, wrote the first draft, and wrote the online documentation; OC, JAM, MK, JAP, and PLG edited and approved the final version of the manuscript.

6. Acknowledgements

This work was supported by the Natural Science and Engineering Council of Canada (RGPIN-2018-05458 to PLG and RGPIN-2022-04421 to JAP) and the Canadian Institutes of Health Research (PJT-156241 to PLG, PJT-175010 to JAP). JAM was supported by a Banting Postdoctoral Fellowship, a BrainsCAN Postdoctoral Fellowship, and a Vector Institute Postgraduate Affiliate Program Stipend.

7. Competing interests

The authors declare no competing financial or non-financial interests.

8. References

Abbott, L., & Svoboda, K. (2020). Brain-wide interactions between neural circuits. *Current Opinion in Neurobiology*, *65*, iii–v. https://doi.org/10.1016/j.conb.2020.12.012

Bhushan, N., & Shadmehr, R. (1999). Computational nature of human adaptive control during learning of reaching movements in force fields. *Biological Cybernetics*, *81*(1), 39–60. https://doi.org/10.1007/s004220050543

Blum, K. P., Campbell, K. S., Horslen, B. C., Nardelli, P., Housley, S. N., Cope, T. C., & Ting, L. H. (2020). Diverse and complex muscle spindle afferent firing properties emerge from multiscale muscle mechanics. *eLife*, *9*, e55177. https://doi.org/10.7554/eLife.55177

Cheng, E. J., & Scott, S. H. (2000). Morphometry of Macaca mulatta forelimb. I. Shoulder and elbow muscles and segment inertial parameters. *Journal of Morphology*, *245*(3), 206–224. https://doi.org/10.1002/1097-4687(200009)245:3<206::AID-JMOR3>3.0.CO;2-U

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation* (arXiv:1406.1078). arXiv. http://arxiv.org/abs/1406.1078

Churchland, M. M., & Shenoy, K. V. (2007). Temporal Complexity and Heterogeneity of Single-Neuron Activity in Premotor and Motor Cortex. *Journal of Neurophysiology*, *97*(6), 4235–4257. https://doi.org/10.1152/jn.00095.2007

Cisek, P. (2019). Resynthesizing behavior through phylogenetic refinement. *Attention, Perception, & Psychophysics*, *81*(7), 2265–2287. https://doi.org/10.3758/s13414-019-01760-1

Conditt, M. A., Gandolfo, F., & Mussa-Ivaldi, F. A. (1997). The motor system does not learn the dynamics of the arm by rote memorization of past experience. *Journal of Neurophysiology*, *78*(1), 554–560. https://doi.org/10.1152/jn.1997.78.1.554

Delp, S. L., Anderson, F. C., Arnold, A. S., Loan, P., Habib, A., John, C. T., Guendelman, E., & Thelen, D. G. (2007). OpenSim: Open-Source Software to Create and Analyze Dynamic Simulations of Movement. *IEEE Transactions on Biomedical Engineering*, *54*(11), 1940–1950. https://doi.org/10.1109/TBME.2007.901024

Dimitriou, M., Wolpert, D. M., & Franklin, D. W. (2013). The Temporal Evolution of Feedback Gains Rapidly Update to Task Demands. *Journal of Neuroscience*, *33*(26), 10898–10909. https://doi.org/10.1523/JNEUROSCI.5669-12.2013

Driscoll, L., Shenoy, K., & Sussillo, D. (2022). *Flexible multitask computation in recurrent networks utilizes shared dynamical motifs* [Preprint]. Neuroscience. https://doi.org/10.1101/2022.08.15.503870

Feldman, A. G., & Levin, M. F. (1995). The origin and use of positional frames of reference in motor control. *Behavioral and Brain Sciences*, *18*(4), 723–744. https://doi.org/10.1017/S0140525X0004070X

Fetz, E. E. (1993). Dynamic recurrent neural network models of sensorimotor behavior. In *The neurobiology of Neural Networks* (Cambridge MA, pp. 165–190). MIT Press.

Flanagan, J. R., Ostry, D. J., & Feldman, A. G. (1993). Control of Trajectory Modifications in Target-Directed Reaching. *Journal of Motor Behavior*, *25*(3), 140–152. https://doi.org/10.1080/00222895.1993.9942045

Fortunato, C., Bennasar-Vázquez, J., Park, J., Chang, J. C., Miller, L. E., Dudman, J. T., Perich, M. G., & Gallego, J. A. (2023). *Nonlinear manifolds underlie neural population activity during behaviour* (p. 2023.07.18.549575). bioRxiv. https://doi.org/10.1101/2023.07.18.549575

Fujimoto, S., Hoof, H., & Meger, D. (2018). Addressing Function Approximation Error in Actor-Critic Methods. *Proceedings of the 35th International Conference on Machine Learning*, 1587–1596. https://proceedings.mlr.press/v80/fujimoto18a.html

Gershman, S. J., & Ölveczky, B. P. (2020). The neurobiology of deep reinforcement learning. *Current Biology*, *30*(11), R629–R632. https://doi.org/10.1016/j.cub.2020.04.021

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 249–256. https://proceedings.mlr.press/v9/glorot10a.html

Gomi, H., & Kawato, M. (1993). Neural network control for a closed-loop System using Feedback-error-learning. *Neural Networks*, *6*(7), 933–946. https://doi.org/10.1016/S0893-6080(09)80004-X

Gomi, H., & Kawato, M. (1997). Human arm stiffness and equilibrium-point trajectory during multi-joint movement. *Biological Cybernetics*, *76*(3), 163–171. https://doi.org/10.1007/s004220050329

Gribble, P. L., & Ostry, D. J. (2000). Compensation for loads during arm movements using equilibrium-point control. *Experimental Brain Research*, *135*(4), 474–482. https://doi.org/10.1007/s002210000547

Hu, W., Xiao, L., & Pennington, J. (2020). *Provable Benefit of Orthogonal Initialization in Optimizing Deep Linear Networks* (arXiv:2001.05992). arXiv. https://doi.org/10.48550/arXiv.2001.05992

Jordan, M. I., & Rumelhart, D. E. (1992). Forward Models: Supervised Learning with a Distal Teacher. *Cognitive Science*, *16*(3), 307–354. https://doi.org/10.1207/s15516709cog1603_1

Keeley, S. L., Zoltowski, D. M., Aoi, M. C., & Pillow, J. W. (2020). Modeling statistical dependencies in multi-region spike train data. *Current Opinion in Neurobiology*, *65*, 194–202. https://doi.org/10.1016/j.conb.2020.11.005

Kistemaker, D. A., Van Soest, A. (Knoek) J., & Bobbert, M. F. (2006). Is Equilibrium Point Control Feasible for Fast Goal-Directed Single-Joint Movements? *Journal of Neurophysiology*, *95*(5), 2898–2912. https://doi.org/10.1152/jn.00983.2005

Kistemaker, D. A., Wong, J. D., & Gribble, P. L. (2010). The Central Nervous System Does Not Minimize Energy Cost in Arm Movements. *Journal of Neurophysiology*, *104*(6), 2985–2994. https://doi.org/10.1152/jn.00483.2010

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), Article 7553. https://doi.org/10.1038/nature14539

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2019). Continuous control with deep reinforcement learning. *arXiv*. http://arxiv.org/abs/1509.02971

Lillicrap, T. P., Santoro, A., Marris, L., Akerman, C. J., & Hinton, G. (2020). Backpropagation and the brain. *Nature Reviews Neuroscience*, *21*(6), 335–346. https://doi.org/10.1038/s41583-020-0277-3

Lillicrap, T. P., & Scott, S. H. (2013). Preference Distributions of Primary Motor Cortex Neurons Reflect Control Solutions Optimized for Limb Biomechanics. *Neuron*, *77*(1), 168–179. https://doi.org/10.1016/j.neuron.2012.10.041

Lindsay, G. W. (2021). *Models of the Mind: How Physics, Engineering and Mathematics Have Shaped Our Understanding of the Brain*. Bloomsbury Sigma.

Loeb, G. E. (2021). Learning to Use Muscles. *Journal of Human Kinetics*, *76*(1), 9–33. https://doi.org/10.2478/hukin-2020-0084

Michaels, J. A., Dann, B., & Scherberger, H. (2016). Neural Population Dynamics during Reaching Are Better Explained by a Dynamical System than Representational Tuning. *PLOS Computational Biology*, *12*(11), e1005175. https://doi.org/10.1371/journal.pcbi.1005175

Michaels, J. A., Schaffelhofer, S., Agudelo-Toro, A., & Scherberger, H. (2020). A goal-driven modular neural network predicts parietofrontal neural dynamics during grasping. *Proceedings of the National Academy of Sciences*, *117*(50), 32124–32135. https://doi.org/10.1073/pnas.2005087117

Millard, M., Uchida, T., Seth, A., & Delp, S. L. (2013). Flexing Computational Muscle: Modeling and Simulation of Musculotendon Dynamics. *Journal of Biomechanical Engineering*, *135*(2), 021005. https://doi.org/10.1115/1.4023390

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533. https://doi.org/10.1038/nature14236

Murray, W. M., Delp, S. L., & Buchanan, T. S. (1995). Variation of muscle moment arms with elbow and forearm position. *Journal of Biomechanics*, *28*(5), 513–525. https://doi.org/10.1016/0021-9290(94)00114-J

Nijhof, E.-J., & Kouwenhoven, E. (2000). Simulation of Multijoint Arm Movements. In J. M. Winters & P. E. Crago (Eds.), *Biomechanics and Neural Control of Posture and Movement* (pp. 363–372). Springer New York. https://doi.org/10.1007/978-1-4612-2104-3_29

Pesaran, B., Hagan, M., Qiao, S., & Shewcraft, R. (2021). Multiregional communication and the channel modulation hypothesis. *Current Opinion in Neurobiology*, *66*, 250–257. https://doi.org/10.1016/j.conb.2020.11.016

Pruszynski, J. A., King, G. L., Boisse, L., Scott, S. H., Flanagan, J. R., & Munoz, D. P. (2010). Stimulus-locked responses on human arm muscles reveal a rapid neural pathway linking visual input to arm motor output. *The European Journal of Neuroscience*, *32*(6), 1049–1057. https://doi.org/10.1111/j.1460-9568.2010.07380.x

Pruszynski, J. A., Omrani, M., & Scott, S. H. (2014). Goal-Dependent Modulation of Fast Feedback Responses in Primary Motor Cortex. *Journal of Neuroscience*, *34*(13), 4608–4617. https://doi.org/10.1523/JNEUROSCI.4520-13.2014

Reschechtko, S., & Pruszynski, J. A. (2020). Stretch reflexes. *Current Biology*, *30*(18), R1025–R1030. https://doi.org/10.1016/j.cub.2020.07.092

Richards, B. A., Lillicrap, T. P., Beaudoin, P., Bengio, Y., Bogacz, R., Christensen, A., Clopath, C., Costa, R. P., de Berker, A., Ganguli, S., Gillon, C. J., Hafner, D., Kepecs, A., Kriegeskorte, N., Latham, P., Lindsay, G. W., Miller, K. D., Naud, R., Pack, C. C., … Kording, K. P. (2019). A deep learning framework for neuroscience. *Nature Neuroscience*, *22*(11), Article 11. https://doi.org/10.1038/s41593-019-0520-2

Rumelhart, D. E., Hintont, G. E., & Williams, R. J. (1986). *Learning representations by back-propagating errors*. 4.

Safaie, M., Chang, J. C., Park, J., Miller, L. E., Dudman, J. T., Perich, M. G., & Gallego, J. A. (2023). Preserved neural dynamics across animals performing similar behaviour. *Nature*, *623*(7988), Article 7988. https://doi.org/10.1038/s41586-023-06714-0

Saxe, A., Nelli, S., & Summerfield, C. (2021). If deep learning is the answer, what is the question? *Nature Reviews Neuroscience*, *22*(1), 55–67. https://doi.org/10.1038/s41583-020-00395-8

Scott, S. H., Gribble, P. L., Graham, K. M., & Cabel, D. W. (2001). Dissociation between hand motion and population vectors from neural activity in motor cortex. *Nature*, *413*(6852), Article 6852. https://doi.org/10.1038/35093102

Scott, S. H., & Kalaska, J. F. (1997). Reaching Movements With Similar Hand Paths But Different Arm Orientations. I. Activity of Individual Cells in Motor Cortex. *Journal of Neurophysiology*, *77*(2), 826–852. https://doi.org/10.1152/jn.1997.77.2.826

Semedo, J. D., Gokcen, E., Machens, C. K., Kohn, A., & Yu, B. M. (2020). Statistical methods for dissecting interactions between brain areas. *Current Opinion in Neurobiology*, *65*, 59–69. https://doi.org/10.1016/j.conb.2020.09.009

Seth, A., Hicks, J. L., Uchida, T. K., Habib, A., Dembia, C. L., Dunne, J. J., Ong, C. F., DeMers, M. S., Rajagopal, A., Millard, M., Hamner, S. R., Arnold, E. M., Yong, J. R., Lakshmikanth, S. K., Sherman, M. A., Ku, J. P., & Delp, S. L. (2018). OpenSim: Simulating musculoskeletal dynamics and neuromuscular control to study human and animal movement. *PLOS Computational Biology*, *14*(7), e1006223. https://doi.org/10.1371/journal.pcbi.1006223

Seth, A., Sherman, M., Eastman, P., & Delp, S. (2010). Minimal formulation of joint motion for biomechanisms. *Nonlinear Dynamics*, *62*(1–2), 291–303. https://doi.org/10.1007/s11071-010-9717-3

Shadmehr, R., & Krakauer, J. W. (2008). A computational neuroanatomy for motor control. *Experimental Brain Research*, *185*(3), 359–381. https://doi.org/10.1007/s00221-008-1280-5

Shadmehr, R., & Mussa-Ivaldi, F. A. (1994). Adaptive representation of dynamics during learning of a motor task. *Journal of Neuroscience*, *14*(5), 3208–3224. https://doi.org/10.1523/JNEUROSCI.14-05-03208.1994

Sherman, M. A., Seth, A., & Delp, S. L. (2013). What is a Moment Arm? Calculating Muscle Effectiveness in Biomechanical Models Using Generalized Coordinates. *Volume 7B: 9th International Conference on Multibody Systems, Nonlinear Dynamics, and Control*, V07BT10A052. https://doi.org/10.1115/DETC2013-13633

Thelen, D. G. (2003). Adjustment of Muscle Mechanics Model Parameters to Simulate Dynamic Contractions in Older Adults. *Journal of Biomechanical Engineering*, *125*(1), 70–77. https://doi.org/10.1115/1.1531112

Todorov, E. (2004). Optimality principles in sensorimotor control. *Nature Neuroscience*, *7*(9), 907–915. https://doi.org/10.1038/nn1309

Todorov, E., Erez, T., & Tassa, Y. (2012). MuJoCo: A physics engine for model-based control. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 5026–5033. https://doi.org/10.1109/IROS.2012.6386109

Towers, M., Terry, J. K., Kwiatkowski, A., Balis, J. U., Cola, G., Deleu, T., Goulão, M., Kallinteris, A., KG, A., Krimmel, M., Perez-Vicente, R., Pierré, A., Schulhoff, S., Tai, J. J., Tan, A. J. S., & Younis, O. G. (2023). *Gymnasium* [Computer software]. Zenodo. https://doi.org/10.5281/zenodo.8148697

Weiler, J., Gribble, P. L., & Pruszynski, J. A. (2019). Spinal stretch reflexes support efficient hand control. *Nature Neuroscience*, *22*(4), 529–533. https://doi.org/10.1038/s41593-019-0336-0

Whittington, J. C. R., & Bogacz, R. (2017). An Approximation of the Error Backpropagation Algorithm in a Predictive Coding Network with Local Hebbian Synaptic Plasticity. *Neural Computation*, *29*(5), 1229–1262. https://doi.org/10.1162/NECO_a_00949

Wierzbicka, M. M., Wiegner, A. W., & Shahani, B. T. (1986). Role of agonist and antagonist muscles in fast arm movements in man. *Experimental Brain Research*, *63*(2), 331–340. https://doi.org/10.1007/BF00236850

Willett, F., Vyas, S., Michaels, J. A., Henderson, J. M., & Shenoy, K. V. (2021, November 9). *Feedback control dynamics explain motor cortical activity*. 50th Annual Meeting of the Society for Neuroscience. https://www.abstractsonline.com/pp8/#!/10485/presentation/willett/1

Won, J., & Hogan, N. (1995). Stability properties of human reaching movements. *Experimental Brain Research*, *107*(1). https://doi.org/10.1007/BF00228024

Zahalak, G. I. (1981). A distribution-moment approximation for kinetic theories of muscular contraction. *Mathematical Biosciences*, *55*(1), 89–114. https://doi.org/10.1016/0025-5564(81)90014-6