A Details of the training procedure

To avoid local minima with narrow spikes of velocity but low overall length, we validate the result during training based on the maximum velocity of Eq. (11) and the path length $L + \lambda_\phi \max_t \phi(t)$, where λ_ϕ is a hyperparameter.

We found that training with batch gradient descent and the loss defined in Eq. (10) is prone to local minima. Therefore, we pre-train the neural network g_ω on n random parametric curves. As random curves we chose Bézier curves [De Casteljau, 1986] of which the control points are obtained as follows: We take $\mathbf{z}_0\tilde{k}/\tilde{K} + \mathbf{z}_1(\tilde{K} - \tilde{k})/\tilde{K},\ \tilde{k} = 1, 2, \dots, \tilde{K} - 1$ as the centers of a uniform distribution, with its support orthogonal to the straight line between $\mathbf{z}_0$ and $\mathbf{z}_1$ and the range $(\mathbf{z}_1 - \mathbf{z}_0)/2$. For each of those random uniforms, we sample once, to obtain a set of $\tilde{K}-1$ random points $\mathbf{z}_{\tilde{k}}$. Together with $\mathbf{z}_0$ and $\mathbf{z}_1$, these define the control points of the Bézier curve. For each of the n random curves, we fit a separate $g_\omega(t)$ to the points of the curve and select the model g_ω with the lowest validation value as the pre-trained model. Afterwards, we proceed with the optimization of the loss Eq. (10).

B Gradients of piecewise linear activation functions

Note that calculating $\partial L(g_\omega(t))/\partial\omega$ involves calculating the gradients of the Jacobian $\partial\mathbf{x}/\partial\mathbf{z}$ as well. Therefore optimization with gradient-based methods is not possible when the generative model uses piecewise linear units. This can be illustrated with an example of a neural network with one hidden layer:

$$\begin{aligned}\frac{\partial \mathbf{J}}{\partial \mathbf{z}} &= \frac{\partial \mathbf{J}}{\partial \mathbf{h}}\frac{\partial \mathbf{h}}{\partial \mathbf{z}} = \frac{\partial}{\partial \mathbf{h}}\Big(\frac{\partial \mathbf{x}}{\partial \mathbf{h}}\frac{\partial \mathbf{h}}{\partial \mathbf{z}}\Big)\frac{\partial \mathbf{h}}{\partial \mathbf{z}} \\ &= \frac{\partial^2 \mathbf{x}}{\partial \mathbf{h}^2}\frac{\partial \mathbf{h}}{\partial \mathbf{z}}\frac{\partial \mathbf{h}}{\partial \mathbf{z}} + \frac{\partial \mathbf{x}}{\partial \mathbf{h}}\frac{\partial^2 \mathbf{h}}{\partial \mathbf{z}\partial \mathbf{h}}\frac{\partial \mathbf{h}}{\partial \mathbf{z}}. \end{aligned} \tag{19}$$

Both terms in Eq. (19) contain a term that involves twice differentiating a layer with an activation function. In the case of piecewise linear units, the derivative is a constant and hence the second differentiation yields zero.

C Gradients of sigmoid, tanh and softplus activation functions

We can easily get the Jacobian using sigmoid, tanh and softplus activation functions. Take one layer with the sigmoid or tanh activation function as an example, the Jacobian is written as

$$\mathbf{J}_i = \frac{\partial x_i}{\partial \mathbf{z}} = x_i(1 - x_i)\mathbf{w}_i, \tag{20}$$

where $\mathbf{w}_i$ is the weights.

With a softplus activation function, the Jacobian is

$$\mathbf{J}_i = \frac{\mathbf{w}_i}{1 + e^{-\mathbf{z}\mathbf{w}_i}}. \tag{21}$$

Consequently, the derivative of Jacobian is straightforward.

D Experiment setups

We used Adam optimizer [Kingma and Ba, 2014] for all experiments. FC in the tables refers to fully-connected layers. In Table 3, for the generative model-architecture we used an MLP and residual connections [He et al., 2016]—additionally to the input and output layer. K is the number of importance-weighted samples in Eq. (3).

Table 1: The setup for geodesic neural networks

architecture	**hyperparameters**
Input $\in \mathbb{R}^{N_t}$	learning rate $= 10^{-2}$
2 tanh FC $\times$ 150 units	500 sample points
Output $\in \mathbb{R}^{N_z}$	

Table 2: The setup for the pendulum dataset

recognition model	**generative model**	**hyperparameters**
Input $\in \mathbb{R}^{256}$	Input $\in \mathbb{R}^2$	learning rate $= 10^{-4}$
2 tanh FC $\times$ 512 units	2 tanh FC $\times$ 512 units	$K = 50$
linear FC output layer for means	softplus FC output layer for means	batch size $= 20$
softplus FC output layer for variances	global variable for variances	

Table 3: The setup for the MNIST dataset

recognition model	**generative model**	**hyperparameters**
Input $\in \mathbb{R}^{784}$	Input $\in \mathbb{R}^2$	learning rate $= 10^{-4}$
2 tanh FC $\times$ 512 units	7 residual $\times$ 128 units.	$K = 50$
linear FC output layer for means	softplus FC output layer for means	batch size $= 20$
softplus FC output layer for variances	global variable for variances	

Table 4: The setup for the robot arm simulaiton dataset

recognition model	**generative model**	**hyperparameters**
Input $\in \mathbb{R}^6$	Input $\in \mathbb{R}^2$	learning rate $= 10^{-3}$
2 tanh FC $\times$ 512 units	2 tanh FC $\times$ 512 units	$K = 15$
linear FC output layer for means	softplus FC output layer for means	batch size $= 150$
softplus FC output layer for variances	global variable for variances	

Table 5: The setup for the human motion dataset

recognition model	**generative model**	**hyperparameters**
Input $\in \mathbb{R}^{50}$	Input $\in \mathbb{R}^2$	learning rate $= 10^{-3}$
3 tanh FC $\times$ 512 units	3 tanh FC $\times$ 512 units	$K = 15$
linear FC output layer for means	softplus FC output layer for means	batch size $= 150$
softplus FC output layer for variances	global variable for variances	