ARTICLES
https://doi.org/10.1038/s42256-020-00237-3

nature machine intelligence

Neural circuit policies enabling auditable autonomy

Mathias Lechner[1,4] ✉, Ramin Hasani[2,3,4] ✉, Alexander Amini[3], Thomas A. Henzinger[1], Daniela Rus[3] and Radu Grosu[2]

A central goal of artificial intelligence in high-stakes decision-making applications is to design a single algorithm that simultaneously expresses generalizability by learning coherent representations of their world and interpretable explanations of its dynamics. Here, we combine brain-inspired neural computation principles and scalable deep learning architectures to design compact neural controllers for task-specific compartments of a full-stack autonomous vehicle control system. We discover that a single algorithm with 19 control neurons, connecting 32 encapsulated input features to outputs by 253 synapses, learns to map high-dimensional inputs into steering commands. This system shows superior generalizability, interpretability and robustness compared with orders-of-magnitude larger black-box learning systems. The obtained neural agents enable high-fidelity autonomy for task-specific parts of a complex autonomous system.

We set out to design a brain-inspired intelligent agent that learns to control an autonomous vehicle directly from its camera inputs (end-to-end learning to control[1,2]). The agent has to learn a coherent representation of its world from multidimensional sensory information, and utilize it to generalize well in unseen situations. Surprisingly, animals as small as the nematode *Caenorhabditis elegans* have mastered such an ability, to perform locomotion[3], motor control[4] and navigation[5], through their near-optimal nervous system structure[6,7] and their harmonious neural information-processing mechanisms[8]. In complex real-world scenarios, for instance, autonomous driving, such neural computation inspiration[9,10] can lead to more expressive artificial intelligence agents with models that are simultaneously accurate and explainable[11].

Although deep learning algorithms have achieved noteworthy successes in various high-dimensional tasks[2,12–16], there still are important representation-learning challenges[17–19] that have to be addressed. For instance, the domain of end-to-end control is safety critical[20]. This demands interpretable dynamics of the intelligent controllers, as a first step towards investigating their safety issues. Furthermore, while learned vehicle control agents often show great performance in offline testing and simulations, this considerably degrades during live driving. In addition, it is desirable that agents learn the true causal structure[21,22] between the observed driving scenes and their corresponding optimal-steering commands (the specific task of the agent). Ideally, for a lane-keeping task, we wish that the agent implicitly learns to attend to the road's horizon when taking a current steering decision, while maintaining an attractive performance on short-term steering. However, in practice, performant models have been shown to learn a variety of unfair[23] and suboptimal[22] input–output causal structures[24,25]. Finally, within the processing pipeline of the high-dimensional data-stream input, the agent has to incorporate a short-term memory mechanism capturing temporal dependencies.

The successful end-to-end autonomous-control approaches to lane-keeping[2,26–28] (Fig. 1) rely solely on deep convolutional neural network architectures[29], steering a vehicle at a time *t*, based on the most recent camera frame[30] (Fig. 2a). While such feedforward models can properly drive the vehicle in case of ideal input data, they often fail if the data are noisy. This is because they do not exploit the temporal nature of the task, enabling them to filter out transient disturbances. As a result, temporary corruptions of the input stream (that is, sudden sunlight, as illustrated in Fig. 2a) lead to unstable predictions. On the contrary, recurrent neural networks (RNNs)[31,32] are a class of artificial neural networks that take into account past observations at a current output decision, through a feedback mechanism. Thus, in principle, they should lead to more robust end-to-end controllers (Fig. 2b). RNNs are trained over finite-length labelled training sequences by the backpropagation algorithm[33] applied to their unfolded feedforward representation[32] (Figs. 2c,d). Historically, training RNNs has been challenging due to their elevated or vanishing gradients during the learning phase[31,32]. Owing to the development of advanced, gated RNNs, such as the long short-term memory (LSTM)[34], the challenge is tackled by enforcing a constant error flow, through the fixation of the recurrent weights to 1 and removing nonlinearities within the feedback path[31].

From a time-series-modelling point of view, having a constant error flow is a desirable property, as arbitrary data sequences may have long-term relations (Fig. 2d, right). However, in the case of end-to-end autonomous driving, learning long-term dependencies can be detrimental, due to the short-term causality of the underlying task. When driving a vehicle to follow the lane, humans do not recall images of the road from more than a few seconds ago to operate the steering wheel[35]. Consequently, LSTM networks may capture spurious long-term dependencies that may have been present in the training data, and thus learn inadequate causal models[21]. On the contrary, vanishing of gradients prevents RNNs from learning correlations of events with long-term-dependencies[36–38]. This property counterintuitively enhances the real-world control performance of a learned RNN agent, as it places a prior on the temporal attention span of the network, to the most recent few observations.

[1]Institute of Science and Technology Austria (IST Austria), Klosterneuburg, Austria. [2]Technische Universität Wien (TU Wien), Vienna, Austria. [3]Massachusetts Institute of Technology (MIT), Cambridge, USA. [4]These authors contributed equally: Mathias Lechner, Ramin Hasani.
✉e-mail: mathias.lechner@ist.ac.at; rhasani@mit.edu

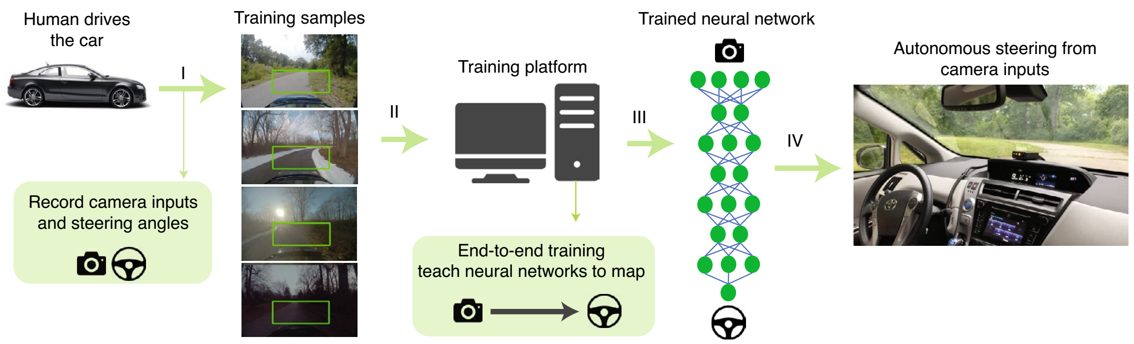

Human drives the car
Training samples
Training platform
Trained neural network
Autonomous steering from camera inputs
I
II
III
IV
Record camera inputs and steering angles
End-to-end training teach neural networks to map

Fig. 1 | End-to-end driving. The process starts by collecting a considerable amount of human driving experiences, in a car that is equipped with camera(s) and in-car computing units. The diverse set of training samples are then edited (green boxes) and are labelled by their corresponding steering angle. An end-to-end training algorithm trains and validates an artificial neural network agent, in a supervised learning fashion to directly turn camera inputs into steering decisions. The obtained network is then deployed on the high-performance computing units mounted inside the car to drive the car autonomously in real unseen environments.

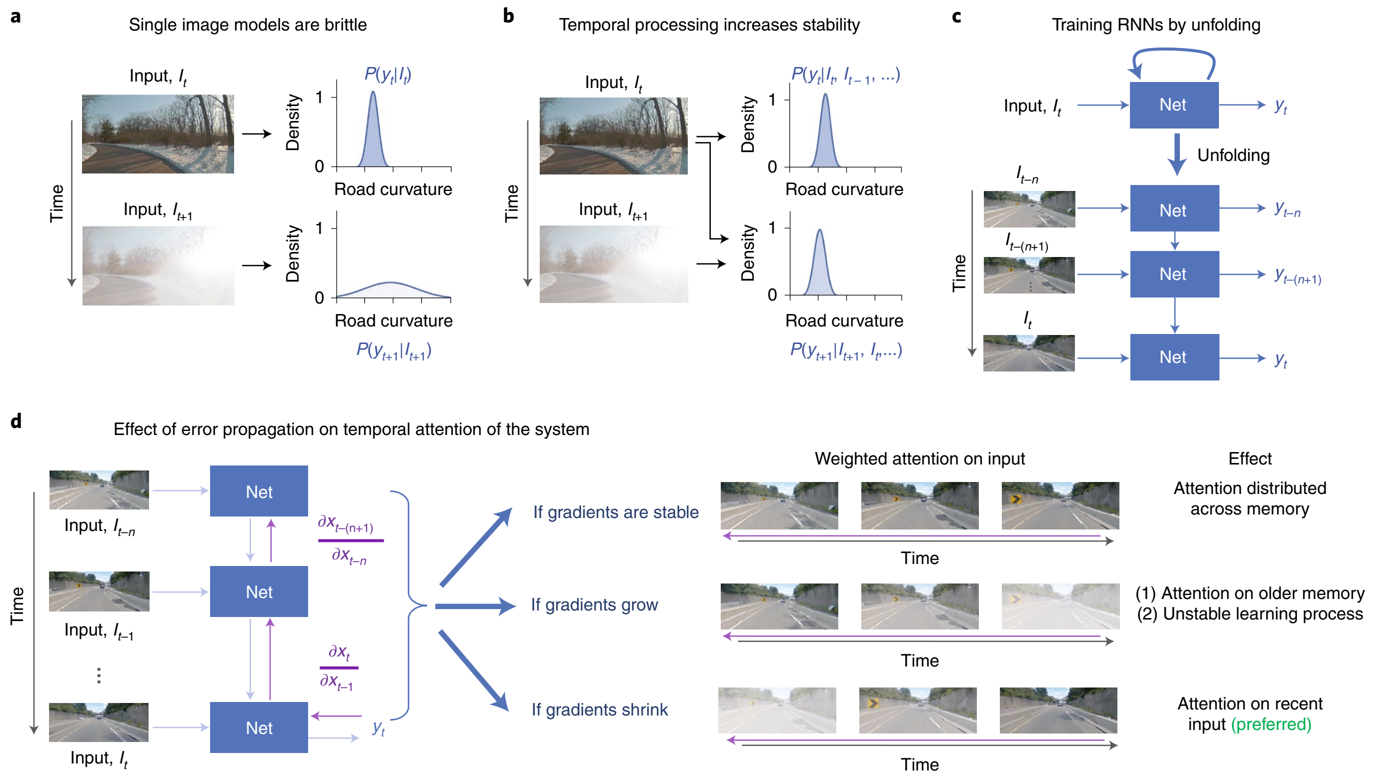

a
Single image models are brittle
Input, I_t
$P(y_t|I_t)$
Input, I_{t+1}
$P(y_{t+1}|I_{t+1})$
Density
Road curvature
Time
b
Temporal processing increases stability
$P(y_t|I_t, I_{t-1}, ...)$
$P(y_{t+1}|I_{t+1}, I_t, ...)$
c
Training RNNs by unfolding
Net
y_t
Unfolding
I_{t-n}
y_{t-n}
$I_{t-(n+1)}$
$y_{t-(n+1)}$
d
Effect of error propagation on temporal attention of the system
Input, I_{t-n}
Input, I_{t-1}
Input, I_t
$\frac{\partial x_{t-(n+1)}}{\partial x_{t-n}}$
$\frac{\partial x_t}{\partial x_{t-1}}$
If gradients are stable
If gradients grow
If gradients shrink
Weighted attention on input
Effect
Attention distributed across memory
(1) Attention on older memory
(2) Unstable learning process
Attention on recent input (preferred)

Fig. 2 | Recurrent network modules are essential for the lane-keeping tasks. a, A feedforward CNN network computes its output, $P(y_t|I_t)$, by relying solely on the current observation, I_t. Consequently, inputs that are corrupted by transient perturbations (bottom) will result in high output variance and faulty decisions. **b**, An RNN has access to past observations at a current driving step, enabling it to filter out transient corruptions that are present in the input stream. **c**, Training RNNs by unrolling their state in time. **d**, Then, applying backpropagation through time in an unfolded RNN. Purple derivatives indicate the dependency of the loss function's derivative with respect to an RNN's state weights to the evolution of the RNN's state, $x(t)$, in time. Blurred images depict weaker attention of the RNN when computing a current decision. n is the number of unfolding steps.

The development of a single, task-specific algorithm that universally satisfies the representation-learning challenges described above has been a central goal of artificial intelligence[9,10]. To advance towards this goal, we draw inspiration from the neural computations known to happen in biological brains[6,7,39,40] and achieve a remarkable degree of controllability[3–5,8]. We develop compact representations called neural circuit policies (NCPs), where each neuron has increased computational capabilities[41] compared with contemporary deep models. We show that NCPs lead to sparse networks that are more easily interpretable and demonstrate this in the context of autonomous driving. We discovered that for the lane-keeping task mentioned above, very small networks of brain-inspired neural models (that is, networks with a control compartment consisting of only 19 neurons), in combination with compact convolutional neural networks (CNNs)[29], achieved superior performance, compared with state-of-the-art models, in learning how to steer a

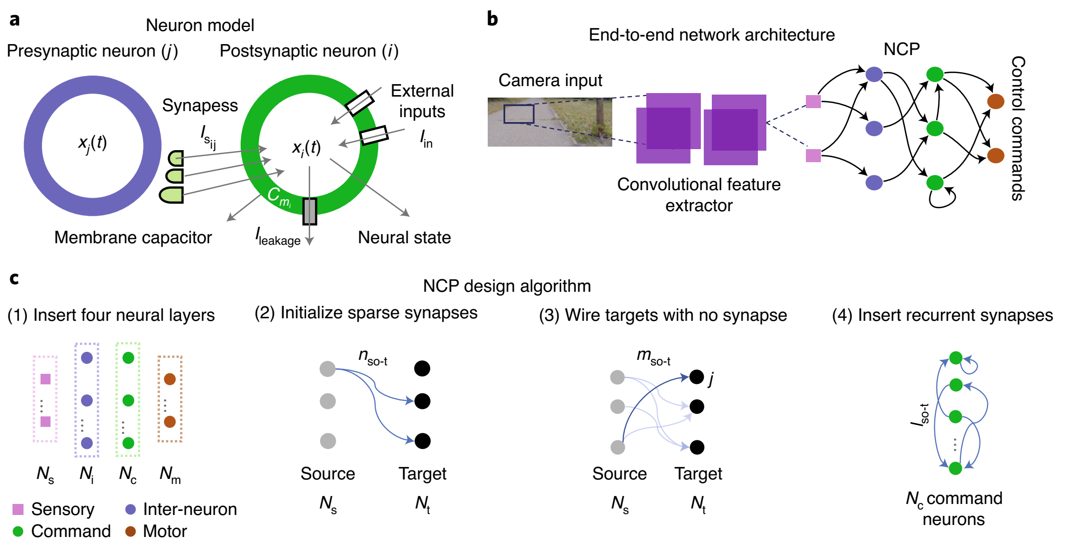


Fig. 3 | Designing NCP networks with an LTC neural model. a, Representation of the neural state, $x_i(t)$, of a postsynaptic LTC neuron i receiving input currents from presynaptic neuron, j. The neural state is determined by the aggregation of the inflows/outflows to/from the cell. I_{in} is the external input currents, $I_{leakage}$ is the leakage current. Synaptic currents ($I_{s_{ij}}$) are set by an input dependent nonlinearity f that is a function of the presynaptic neural state, $x_j(t)$ and its synaptic parameters (see Methods for further details). **b**, Representation of an NCP end-to-end network; it perceives the camera inputs that are transformed by a set of convolutional layers to a latent representation, which is exploited by the designed NCP (based on the steps described in **c**) to produce control actions, to command control orders. **c**, NCP design procedure based on rules 1 to 4 in the main text (see algorithms 2 to 6 in Methods).

vehicle directly from high-dimensional inputs. Here we use the representation-learning challenges as the main criteria for assessing the performance of autonomous-control agents.

Designing and learning NCPs

To address the representation-learning challenges and the complexity of autonomous lane-keeping, we design an end-to-end learning system that perceives the inputs by a set of convolutional layers[42], extracts image features and performs control by an RNN structure, termed an NCP.

The network structure of NCPs is inspired by the wiring diagram of the *C. elegans* nematode[43]. Many neural circuits within the nematode's nervous system are constructed by a distinct four-layer hierarchical network topology. They receive environmental observations through sensory neurons. These are passed on to inter-neurons and command neurons, which generate an output decision. Finally, this decision is passed to the motor neurons to actuate its muscles. The wiring diagram of *C. elegans* achieves a sparsity of around 90% (ref. [6]), with predominantly feedforward connections from sensors to intermediate neurons, highly recurrent connections among inter-neurons and command neurons, and feedforward connections from command neurons to motor neurons. This specific topology was shown to have attractive computational advantages, such as, efficient distributed control, requiring a small number of neurons[6], hierarchical temporal dynamics[8], robot-learning capabilities[44] and maximal information propagation in sparse-flow networks[45].

Neural dynamics of NCPs are given by continuous-time ordinary differential equations (ODEs), originally developed to capture the dynamics of the nervous system of small species, such as *C. elegans*[41] (Fig. 3a). At their core, NCPs possess a nonlinear time-varying synaptic transmission mechanism that improves their expressive power in modelling time series, compared with their deep learning counterparts[41]. The foundational neural building blocks of NCPs are called liquid time constant (LTC) networks [41]. Further details about LTCs are given in Methods.

The architecture of an NCP network is determined by the design principles introduced in rules 1–4, corresponding to the steps presented in Fig. 3c, as follows:

(1) Insert four neural layers—N_s sensory neurons, N_i inter-neurons, N_c command neurons and N_m motor neurons ((1) in Fig. 3c).
(2) Between every two consecutive layers—∀ source neuron, insert n_{so-t} synapses ($n_{so-t} \le N_t$), with synaptic polarity ~Bernoulli(p_2), to n_{so-t} target neurons, randomly selected ~Binomial(n_{so-t}, p_1) ((2) in Fig. 3c). n_{so-t} is the number of synapses from source to target. p_1 and p_2 are probabilities corresponding to their distributions.
(3) Between every two consecutive layers—∀ target neuron j with no synapse, insert m_{so-t} synapses ($m_{so-t} \le \frac{1}{N_t}\sum_{i=1, i\neq j}^{N_t} L_{t_i}$), where L_{t_i} is the number of synapses to target neuron i, with synaptic polarity (being excitatory or inhibitory) ~Bernoulli(p_2), from m_{so-t} source neurons, randomly selected from ~Binomial(m_{so-t}, p_3) ((3) in Fig. 3c). m_{so-t} is the number of synapses from source to target neurons with no synaptic connections.
(4) Recurrent connections of command neurons—∀ command neuron, insert l_{so-t} synapses ($l_{so-t} \le N_c$), with synaptic polarity ~Bernoulli(p_2), to l_{so-t} target command neurons, randomly selected from ~Binomial(l_{so-t}, p_4) ((4) in Fig. 3c). l_{so-t} is the number of synapses from one interneuron to target neurons.

Applying the NCP design principles above results in very compact and sparse networks of LTC neurons (see the NCP design algorithms in Methods). The learning system corresponding to the lane-keeping task consists of the convolutional frontend, stacked with the NCP network (Fig. 3b). This system is trained in an end-to-end, supervised learning fashion. Given a designed NCP network, we apply a semi-implicit ODE solver to obtain a numerically accurate and stable solution of the system[41]. We then recursively fold the ODE solver call, into an RNN cell and prepare the system's training pipeline. Further details on the training setup are provided in Methods. From the gradient propagation perspective, our approach gives rise to a vanishing gradient phenomenon, which, as described in Fig. 2d, is the preferable setting for learning a real-world autonomous vehicle control (see the proof in Methods).

A large-scale selection of labelled training data were collected by recording the observations and actions of a human driver (see Methods for more details). End-to-end driving is a feedback control

Table 1 | Results of the passive lane-keeping tenfold cross-testing evaluation

Model	Training square error	Test squared error
CNN	1.41 ± 0.30	4.28 ± 4.63
Vanilla RNN	0.14 ± 0.05	3.39 ± 4.39
CT-GRU	0.19 ± 0.05	3.63 ± 4.61
CT-RNN (19 units)	0.44 ± 0.14	3.62 ± 4.35
CT-RNN (64 units)	0.23 ± 0.09	3.43 ± 4.55
Sparse CT-RNN (19 units)	0.77 ± 0.35	4.03 ± 4.80
Sparse CT-RNN (64 units)	0.40 ± 0.43	3.72 ± 4.71
GRU	1.25 ± 1.02	5.06 ± 6.64
LSTM (64 units)	0.19 ± 0.05	**3.17** ± 3.85
LSTM (19 units)	0.16 ± 0.06	3.38 ± 4.48
Sparse LSTM (19 units)	1.05 ± 0.57	3.68 ± 5.21
Sparse LSTM (64 units)	0.29 ± 0.14	3.25 ± 3.93
NCP	0.43 ± 0.26	**3.22** ± 3.92
NCP (randomly wired)	2.12 ± 2.93	5.19 ± 5.43
NCP (fully connected)	2.41 ± 3.44	5.18 ± 4.19

Mean ± standard deviation ($n=10$). Sparse LSTM models are trained with projected gradient descent to enforce a 95% sparsity level. All tested NCP architectures are composed of 19 neurons. Boldface numbers depict best performance. GRU, gated recurrent unit.

problem, where the control actuated by the agent proprioceptively affects future observations. However, during the supervised training phase, this feedback mechanism is utterly disregarded.

We observed that such a train–test discrepancy led to situations where a trained neural network model that performs exceptionally well on the labelled sequences in an offline testing environment (Table 1) fails to steer the car safely in a real testing case. Modern RNNs are particularly vulnerable to these scenarios, as their decision-making process heavily relies on past observations. Hence, to properly assess performance, we chose the architectures that worked well during offline testing and evaluated them actively on a real car. We ran a tenfold cross-testing[46] on 94 minutes of labelled sequences recorded in the Boston metropolitan area (see Methods for more details).

To perform a fair comparison, we equipped all RNN models with the same convolutional head that reduces the dimensionality of the input image to a more compact latent representation to be fed into the RNN compartments. We trained and evaluated the networks with the following architectures: a 64-neuron LSTM, a 64-neuron continuous-time (CT)-RNN and a 19-neuron NCP. Moreover, we compared these recurrent agents with the feedforward CNN model developed in Bokarski et al.[2].

Learning a compact neural representation. A full-stack NCP network is 63 times smaller than the CNN network that established the state-of-the-art of end-to-end driving[2]. Its control network is 970 times sparser than that of LSTM and 241 times sparser than that of CT-RNN. An NCP's RNN compartment possesses 233 times smaller trainable parameter space than that of LSTM, and 59 times lower than CT-RNN. Interestingly, the performance achieved by such a compact neural representation is superior to that of other models in multiple aspects of an ideal autonomous mobile robot controller, described as follows.

Avoiding crashes under increasing input perturbations. Compared with all learning systems under test, NCPs are significantly more robust in avoiding crashes (requiring intervention) that are caused by raising the pixel-wise input perturbations (Fig. 4a). The reason for their noise resiliency is that their CT model serves as a filter (equation (1)). Note that the primary objective of this experiment was to identify how differently each model relies on its memory for making a prediction. As outlined in Fig. 2, an ideal model should incorporate temporal information to allow the filtering of any form of perturbation. To demonstrate this, we used additive zero-mean Gaussian noise, because such noise was not present in the data used for the training process, and it required a minimal assumption on the form, shape and the severity of the perturbation signal. Notwithstanding, we already simulated lens flares during training. Thus, we could expect that all models would tolerate lens flares to some degree.

Robustness of the output decisions in the presence of input noise. Figure 4d,e depicts examples of crash incidents that happened at the locations shown on the map, when the inputs to the networks were heavily perturbed by an input noise. These panels also illustrate how the attention of each intact network is disrupted by the input noise and caused LSTM and CNN networks to drive the vehicle off-road (see Methods 'Saliency map computation'). We quantified the influence of the input perturbations on the attention maps by computing their structural similarity index (SSIM), represented in Fig. 4b. The SSIM indicates how much the structure of the attention maps gets distorted when the incoming inputs are perturbed. The closer SSIM is to 1, the less distorted is the attention. Thus, the network can handle input noise more robustly. The closer the SSIM index is to 0, the more distorted the network's attention is, which results in the increased uncertainty of the network when making a correct driving decision. Under different levels of input perturbations, NCPs consistently maintain a higher SSIM compared with the other models, therefore, reducing its output decision's uncertainty (see Methods 'Structural similarity index').

Driving with smooth neural activity. We quantitatively measured the maximum steepness of the neural dynamics derivative (maximum local Lipschitz constant) for all neurons and report the results in Fig. 4c. We observed that the local decision-making process in NCPs is remarkably smoother than those of other network types. More details are provided in Methods 'Lipschitz continuity computation'.

NCPs enhance interpretability

Interpretation is the process of providing explanations to humans. Although no formal definition for interpretability exists yet[47], we define a model to be more interpretable if its causal mapping between input observations and output decisions, as well as its global hidden-state dynamics up to the cellular level, is more comprehensible to humans[48].

Scalable approaches to interpretability involve the development of algorithms that set quality measures (proxies) for explaining a learning system's dynamics. In particular, for neural network models, numerous works explored post-training qualitative feature visualization techniques for this purpose[49–55]. In addition, computing input-feature attribution measures, such as saliency maps, were effectively used for interpretability[56–59]. Beyond feature visualization, selecting a proper measure for post-training explanations can be arbitrary and challenging[60].

A more systematic way to achieve interpretability would be to design neural architectures that, either through their learning process or through their semantics, result in more transparency[25,61]. However, despite the effectiveness of these approaches for neural networks, the quality of explanations drastically drops as the dimensionality of models increases. This challenge becomes more noticeable when the model architecture is equipped with feedback mechanisms (for example, RNNs).

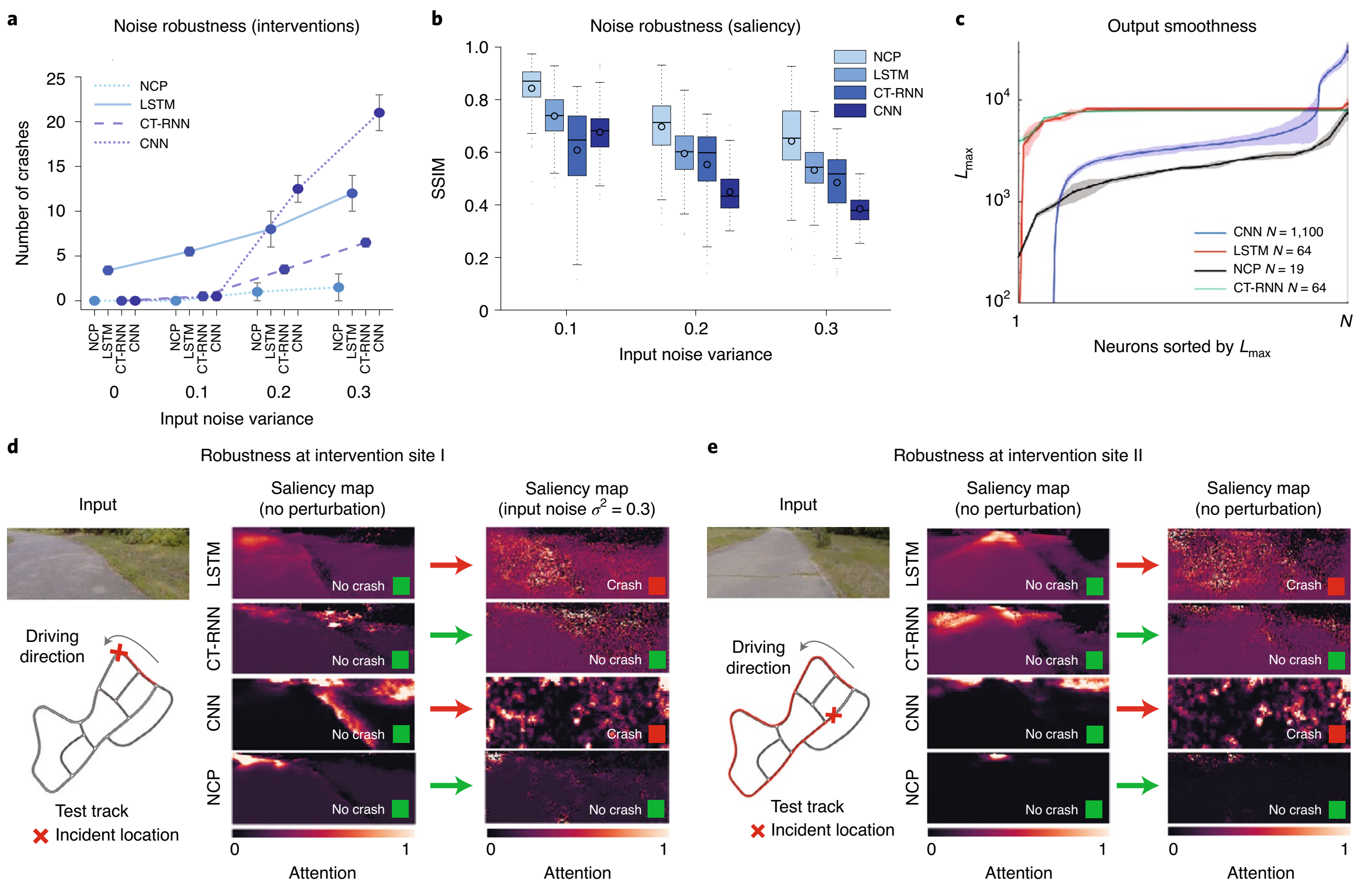


Fig. 4 | Robustness analysis. a, Number of crashes (steering commands with tendency to drive the vehicle off-road) for four networks, as the input noise variance (σ^2) increases, in an active driving test ($n=3$). Additive input noise is sampled from a Gaussian distribution, $\mathcal{N}(0, \sigma^2)$. **b**, Variation of the structural sensitivity index of the saliency maps for four networks as σ^2 increases (see Methods for computational details). A higher value of SSIM is preferable ($n=3$). Each individual box represents the minimum, 25th percentile, median, 75th percentile, interquartile interval, maximum and the outliers. **c**, Maximum Lipschitz constant, L_{max} (an indicator of smoothness and stable dynamics) of the activity of every single neuron (sorted by the amplitude of L_{max} on the horizontal axis) ($n=5$). Lower values are preferable. **d**, Example of the saliency maps before a crash event caused by LSTM and CNN, in the presence of input perturbations, and how it was handled by CT-RNN and NCP. **e**, A second example of a crash incident caused by LSTM and CNN. Error bars in **a** and the cloud intervals in **c** stand for standard deviation.

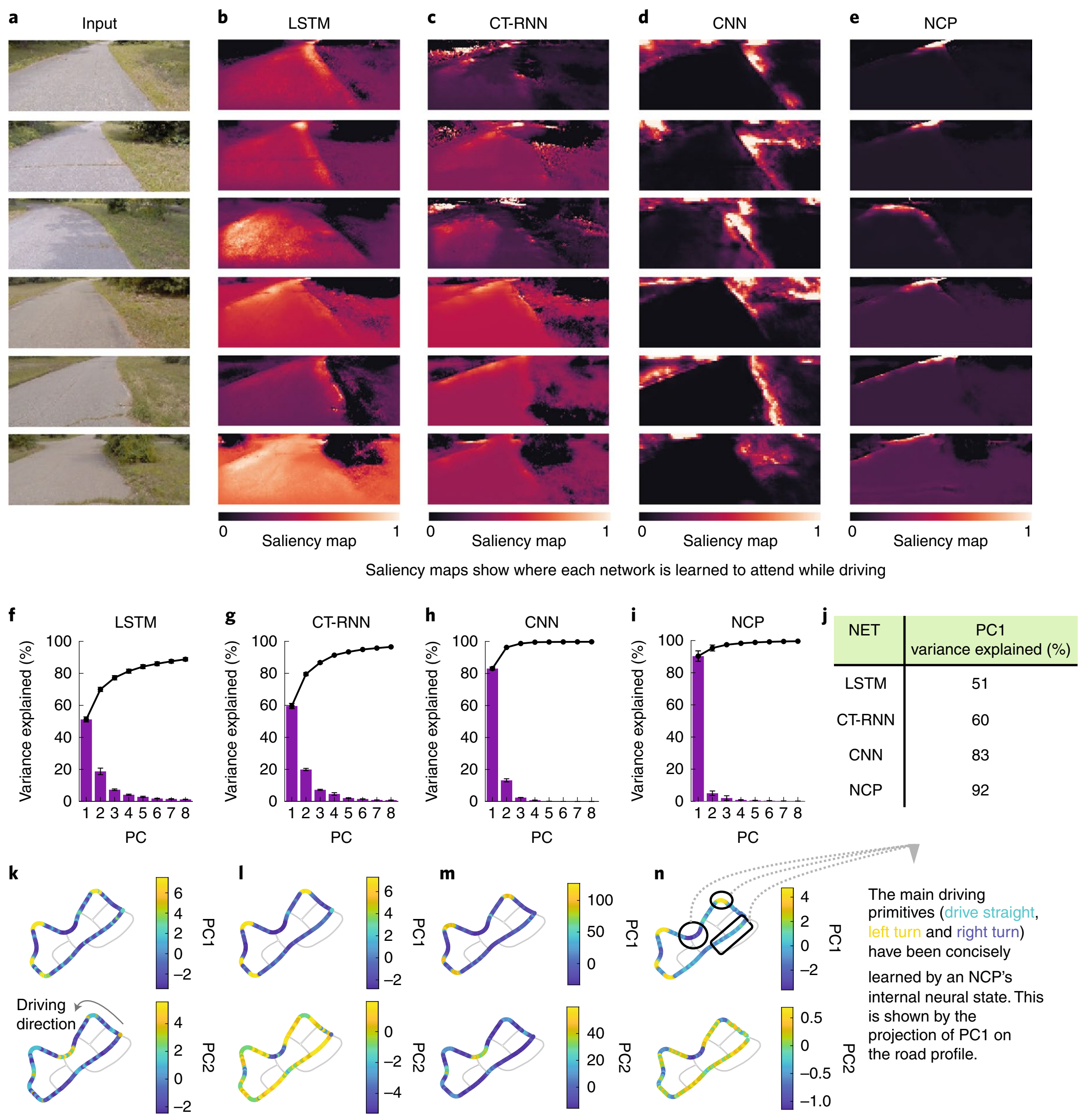
As NCPs realize compact and sparse networks, constructed by expressive neural representations (the LTC model[41]), they ease the interpretation process through known methods, such as saliency map computations[62], dimensionality reduction[63] and cell-contribution analysis[52].

We conduct quantitative interpretability analysis by explaining the attention maps of the convolutional layers, and by computing the global network dynamics of the recurrent network compartment of the models. We then explain cell-level contributions through visualization techniques. For the driving task, we find that there is a close relationship between the geometry of the environment, the specific driving task and the network nodes responsible for the required behaviour. This is a consequence of defining the function of each neuron by differential equations. Accordingly, we experimented with the learned lane-keeping networks to measure their interpretability in three distinct ways.

(1) Explain and visualize where the attention of the convolutional layers is. Figure 5b–e shows sample attention maps of the convolutional parts of the networks during live testing (see Methods 'Saliency map computation'). We have observed that the attention patterns are exclusive to the choice of network architecture (CNN, LSTM, CT-RNN, NCP) and that the explanations are invariant to the choice of hyperparameters (for example, network size).

For instance, the convolutional layers in the NCP networks predominantly attend to the road's horizon to make a driving decision. This is very desirable in the lane-keeping task (Fig. 5e). In contrast, a CNN network looks at the roadside to make a driving decision and ignores the road itself (Fig. 5d). LSTM forced its perception network to learn to attend to the roadside in most scenarios. However, lighting conditions, as well as road profiles, can notably alter the network's attention portfolio (Fig. 5b). The attention of CT-RNN is inconsistent and is heavily influenced by the variations of the road's lighting conditions (Fig. 5c). In the Supplementary Information, we provide an entire collection of saliency maps that we collected during live testing. These maps give an intuitive insight into the decision-making process of a task-specific network within a full-stack autonomous driving system. This insight could help in further safety and robustness analysis.

(2) Global network dynamics. To measure how concisely the networks learned the primitives of driving (straight roads, handling curves and road jitters), we performed a principal component analysis (PCA) and report its variance in Fig. 5f–i. PCA is conducted over the activations of the hidden neurons (without the inclusion of the output signal) of the RNN compartments of the driving networks, collected during live testing. The analysis demonstrated that the first principal component (PC1) of NCP's neural dynamics



NET	PC1 variance explained (%)
LSTM	51
CT-RNN	60
CNN	83
NCP	92

Fig. 5 | Global network dynamics. a, A sequence of input camera images during the active testing. **b–e**, A set of saliency maps computed to obtain the attention of the convolutional layers of the trained networks while driving (see Methods 'Saliency maps computation' for details). **b**, LSTM learned to attend to the roadsides in most scenarios; however, lighting conditions considerably affect its attention portfolio. **c**, CT-RNN's attention is inconsistent and is heavily influenced by the variations of the road's lighting conditions. **d**, CNN learned to drive by focusing on the side roads. **e**, NCP learned to attend to the road's horizon when taking a driving decision. **f–i**, The variance explained by first eight PCs of the activity of all neurons of LSTM (**f**), CT-RNN (**g**), CNN (**h**) and NCP (**i**) ($n=5$). The black line indicates the cumulative variance explained and error bars are standard deviations. **j**, PC1's variance explained for all models. **k–n**, The projection of PC1's (top) and PC2's (bottom) score (the score of the nth PC is computed by $\mathrm{PC}^{(n)}_{\mathrm{score}} = \text{output vector} \times \text{weight}_{\mathrm{PC}^{(n)}}$), over the driving trajectory for LSTM (**k**), CT-RNN (**l**), CNN (**m**) and NCP (**n**). Colour bars depict PC values.

concisely learned the global driving features (explaining 92%), as shown in Fig. 5j, while PC2 learned fine-grained decisions. The conciseness was less apparent in networks with LSTM and CT-RNN recurrent compartments, and therefore it is more challenging to associate their behaviour with intuitive explanations.

To motivate this phenomenon further, we plotted the PC1 and the PC2 scores over the driving trajectory in Fig. 5k–n. NCP is the only model among the others that allocated distinct PC1 activation regimes to the main driving primitives, while fine-grained control decisions have been largely captured by PC2. Other baseline networks require at least two to three PCs to capture the driving profile up to 90%, as shown in Fig. 5f–i. A complementary experiment to further elaborate on this analysis is given in Methods. Consequently, the added value of these results for a more complex autonomous-control system is that the global dynamics of a learned agent can be interpreted and used for further improvements on the task-specific networks (see an additional supporting experiment in Methods and Extended Data Fig. 1).

(3) Cell-level auditability by visualization. The neural state (the amplitude of a neuron's output) and the coupling sensitivity (how a neuron adjusts its reaction speed when interacting with the environment) of LTC cells comprising an NCP network (Fig. 6a) can help

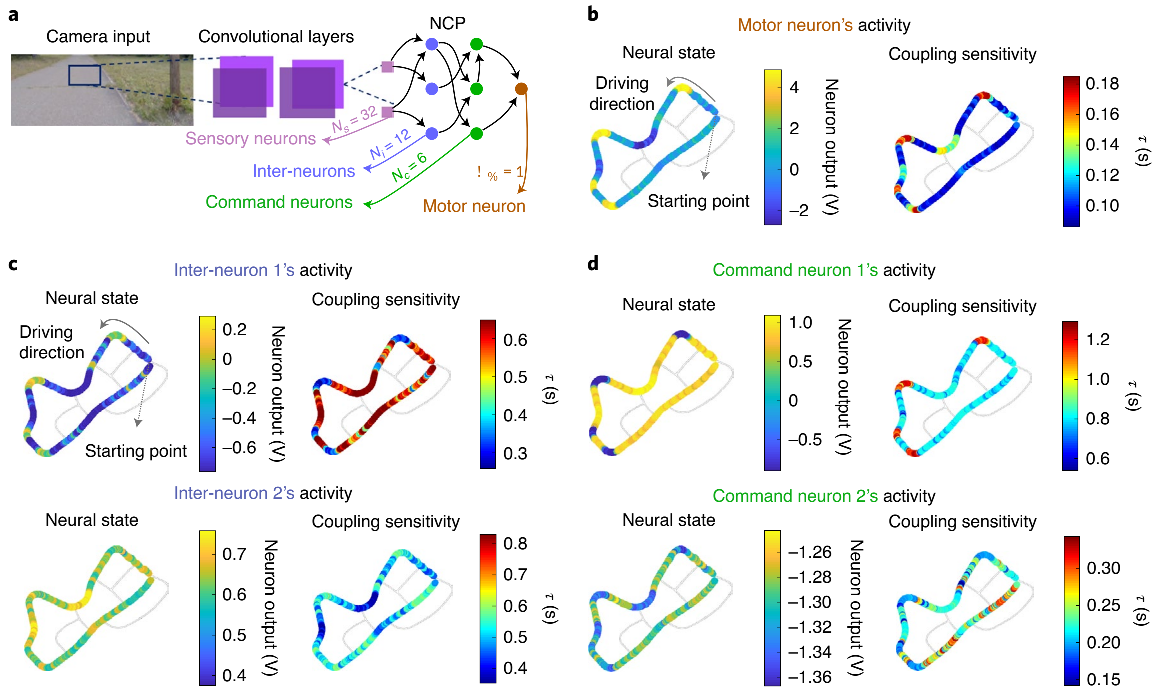


Fig. 6 | Intuitive comprehension of NCP's cells activity while driving. **a**, An NCP network trained end-to-end for autonomous lane-keeping. **b**–**d**, Examples of neural activities projected over the road trajectory on which the car was driven. The neural state (representing the amplitude of a neuron's dynamics) and the coupling sensitivity (representing how a neuron adjusts its reaction speed) are plotted in each subsection. Colour bars on the left plot in each panel depict neuron's output and on the right plot the time constant, τ. **b**, Neural activity of the motor neuron. **c**, Neural activity of two inter-neurons 1 and 2. **d**, Neural activity of two command neurons 1 and 2. An immediate explanation of the cell-level dynamics for the NCP network is achieved and extends to every internal element of the network. See more in Extended Data Figs. 3 and 4.

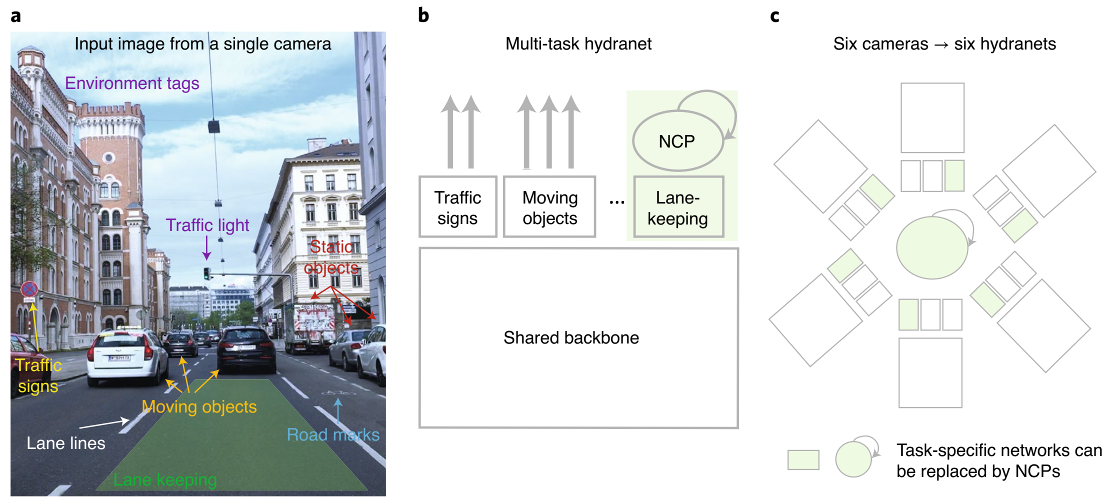


Fig. 7 | NCPs as task-specific networks within a full-stack autonomous vehicle engine. **a**, Camera input and examples of tasks. **b**, An overview of hydranets of Tesla Autopilot[64] redesigned from Tesla at PyTorch 2019[65]. **c**, The overall structure of a vision-based full-stack autonomous vehicle system redesigned from Tesla at PyTorch 2019[65].

to understand how an LTC network's decision is made. Figure 6b–d illustrates the activity of five selected neurons from the NCP driving agent, projected over the driving trajectory. The motor neuron's activity illustrates how the inferred motion primitives correspond to various driving situations (Fig. 6b, left). Its coupling sensitivity demonstrates that the neuron tends to set smoother dynamics while keeping its reaction speed at a relatively constant rate during straight motions. Inter-neuron 1 learned to activate during left turns (Fig. 6c, top left) while adjusting its dynamics to react faster at left-turning events (Fig. 6c, top right). Inter-neuron 2, in contrast, learned to rapidly get more active during right turns (Fig. 6c, bottom). Command neuron 1 is consistently activated during straight driving with a sensitive reaction speed while it is switched off on left turns (Fig. 6d, top). Command neuron 2 is biased at lower membrane potentials and tunes to road jitters when the vehicle drives on a straight path (Fig. 6d, bottom). This degree of immediate interpretation of dynamics is generalizable to every single cell within an NCP.

As the number of computational elements of an NCP system is considerably lower than that of state-of-the-art neural networks, such a degree of access to each cell dynamics could potentially be beneficial for designing fault-test and corner-case analysis to improve the safety of the deployed autonomous system.

NCPs and autonomy

NCPs are highly compact task-specific neural network agents that can proficiently control a vehicle on previously unseen roads, while at the same time being robust to input artefacts, learning short-term causal representations and realizing interpretable dynamics. NCPs are beneficially used within full-stack autonomous vehicle frameworks, given in Fig. 7c. They are designed to improve the performance and transparency of the black-box, task-specific compartments of such complex full-stack autonomous vehicle systems. A vision-based full-stack autopilot has to incorporate many different tasks for the incoming image streams, as shown in Fig. 7a.

State-of-the-art functional autonomous vehicle systems[64] typically share a convolutional backbone network, with many upstream, task-specific networks[65] (Fig. 7b). Although the complexity of the lane-keeping task on which we tested NCPs is relatively low compared with multi-task end-to-end driving, we made sure that NCPs maintain compositionality within full-stack autonomous vehicles, by enabling an end-to-end training pipeline that can backpropagate errors, through the NCPs to the static CNN-based backbone. The resulting task-specific NCPs (for example, for the lane-keeping task) improve many aspects of the contemporary neural control modules in use.

Real-world application domains, such as autonomous driving, avionics, service robots, health and medicine, are surrounded by environmental artefacts and uncertainty, and demand robust real-time decision-making. Moreover, similar to autonomous driving tasks, many applications deal with complex, high-dimensional input–output spaces that become safety critical when deployed in the real world. The success of NCPs in task-specific autonomous vehicles indicates that tackling the complexity of real-world problems does not necessarily require to learn very large neural networks that are hard to comprehend.

Methods

NCP's neural model. An NCP is constructed by a set of LTC neurons, each with state dynamics $x_i(t)$, represented as follows, when connected through an input synapse to a neuron j (ref. [41]):

$$\dot{x}_i = -\left(\frac{1}{\tau_i} + \frac{w_{ij}}{C_{m_i}}\sigma_i(x_j)\right)x_i + \left(\frac{x_{\text{leak}_i}}{\tau_i} + \frac{w_{ij}}{C_{m_i}}\sigma_i(x_j)E_{ij}\right), \tag{1}$$

where $\tau_i = C_{m_i}/g_{l_i}$ is the time constant of the neuron i with a leakage conductance of g_{l_i}, w_{ij} is a synaptic weight from neuron i to j, C_{m_i} is the membrane capacitance, $\sigma_i(x_j(t)) = 1/(1+e^{-\gamma_{ij}(x_j-\mu_{ij})})$, x_{leak_i} is the resting potential and E_{ij} is a reversal synaptic potential that defines the polarity of the synapse. An LTC neuron's overall coupling sensitivity (time constant) is defined by [41]:

$$\tau_{\text{system}_i} = \frac{1}{\frac{1}{\tau_i} + \frac{w_{ij}}{C_{m_i}}\sigma_i(x_j)}. \tag{2}$$

This variable time constant determines the reaction speed of a neuron during decision-making processes, as shown in Fig. 6, for a couple of neurons in the driving NCP. All parameters described in the model are trainable.

Numerical implementation of the NCP networks. To learn the parameters of an NCP circuit, we transform it into a differentiable representation. After modelling the circuit as a system of ordinary differential equations of LTC neurons, we employ a numerical ODE solver to obtain a computable form of it. A solving method suitable for our purpose has to comply with the following three constraints. First, the solver is applied to a real-time system that puts a hard limit on worst-case executing time. Hence, the solver uses a fixed step size[66]. Second, the ODE model of an NCP is stiff[41,66]. Consequently, to avoid numerical instabilities, we adopt a semi-implicit method[66]. Lastly, during the training phase, we compute partial derivatives by backpropagating through the solver. Similar to the stability arguments in the forward path, we need to monitor the error magnitude in the backward phase. In particular, a suitable solving method must not result in an exploding or a rapidly vanishing gradient. To comply with these constraints, we employed a simple Euler approach. As a result, in summary, for each neuron, we adopt a semi-implicit Euler approach with a fixed step size, Δ, of the form:

$$x_i(t+\Delta) := \frac{x_i(t)C_{m_i}/\Delta + g_{l_i}x_{\text{leak}_i}\sum_{j\in I_{\text{in}}} w_{ij}\,\sigma_i(x_j(t))E_{ij}}{C_{m_i}/\Delta + g_{l_i} + \sum_{j\in I_{\text{in}}} w_{ij}\,\sigma_i(x_j(t))}, \tag{3}$$

The set I_{in} represents the set of neurons that are presynaptic to neuron i. This equation was derived from the basic Euler formula[66]:

$$x(t+\Delta) := x(t) + \Delta . f(x(t+\tau), u(t+1)) \tag{4}$$

by setting $\tau = \Delta$ for all $x(t+\tau)$ that appear linear in the nonlinearity f, and setting $\tau = 0$ for all other occurrences (u is the set of inputs to the cell). Note that the well-known explicit (forward) Euler method can be obtained from equation (4) by setting $\tau = 0$. Likewise, the implicit (backward) Euler method is realized by equation (4) if setting $\tau = \Delta$ and solving the resulting nonlinear equation for $x(t+\Delta)$. RNNs usually process their incoming input stream at a fixed sampling frequency (for example, 30 Hz in the described end-to-end driving tasks). To achieve a decent precision, a computation complexity trade-off, we simulated the ODE at a frequency, six times higher than the input sampling rate; we packed six ODE solver steps into one RNN step. In both the training and testing phases, we initialized states of the ODE/RNN by zeros.

NCPs express vanishing gradient. Proof: let $x_i(t)$ be the state of an NCP circuit implemented by our hybrid ODE solver in equation (3). We assume that there is no self-connection synapse $i \to i$. Then

$$\frac{\partial x_i(t+\Delta)}{\partial x_i(t)} = \frac{C_{m_i}/\Delta}{C_{m_i}/\Delta + \underbrace{g_{l_i}}_{\geq 0} + \underbrace{\sum_{j\in I_{\text{in}}} w_{ij}\sigma_i(x_j(t))}_{\geq 0}} \tag{5}$$

$$= \frac{C_{m_i}/\Delta}{C_{m_i}/\Delta + \epsilon}, \quad \text{for some } \epsilon \geq 0 \tag{6}$$

$$\leq 1 \tag{7}$$

In equation (5), we see that the magnitude of the vanishing effect depends on the parameters g_{l_i}, w_{ij} and the current synapse activation $\sigma_i(\ldots)$. In the limit for $g_{l_i}, w_{ij} \to 0$, our hybrid solver approaches a constant error flow, similar to that of an LSTM[34].ϵ stands for the positive value of the second term in the denominator of the right-hand side of equation (5).

Vehicle setup. All data used to train networks was collected on a Toyota Prius 2015 V retrofitted with perception sensors (a forward-facing Leopard Imaging LI-AR0231-GMSL camera), inertial measurement unit (Xsens MTi 100-series IMU), GPS and drive-by-wire steering[67]. All data logging was done directly on an NVIDIA Drive PX2, the in-car high-performance computing platform. The IMU was used to record rotation of the vehicle's rigid body frame and thus, compute the curvature of the vehicle's traversed path. Specifically, given a yaw rate γ_t (rad s^{-1}), and the speed of the vehicle, v_t (m s^{-1}), we compute the curvature of the path as:

$$y_t = \frac{1}{r_t} = \frac{\gamma_t}{v_t} \tag{8}$$

where r_t is the radius of the traversing circle. Ultimately, for the networks learned in this paper, we consider the problem of directly learning a control command from the human-traversed road curvature (y_t) instead of the steering wheel angle (α_t). This is because α_t is a nonlinear function of both y_t and v_t and depends on the tyre slip angle, road surface, weather conditions and vehicle dynamics. Hence, simply learning the steering wheel angle (that is, what the human commanded) is not sufficient for autonomous navigation. Instead, we require knowledge of the traversed road curvature (that is, where the human drove). We can compute the steering wheel angle online by using a bicycle model approximation to control the car at the inference time.

$$\alpha_t = K \arctan(V_L\ y_t) \tag{9}$$

where K is the steering ratio (that is, the ratio between steering and tyre angle) and V_L is the vehicle length.

Passive test dataset. For the passive evaluation, we collected approximately five hours of driving data throughout diverse regions of the Boston metropolitan area during dry, wet and snowy weather conditions on the highway, local and residential roads[68]. We processed the data by removing ambiguous segments, such as lane switches, crossings and congestion, from the recordings. We split the data into ten non-overlapping sets of equally sized chunks for the cross-testing procedure. We trained a model on the union of the remaining nine sets for every ten sets, then evaluated the performance of the model on the withhold test set. The number of training epochs was optimized based on a validation set, which we separated from the union of the nine sets before training. In Table 1, we reported the mean and standard deviation over these ten test iterations.

Active test setup. We conducted the active driving experiments on a private road system. To prepare the models, we collected approximately 94 minutes of data by manoeuvring the vehicle through the test track. We split the data into a training and a validation set of ratio 3:1. The number of training epochs was selected based on the lowest error on the validation set achieved during training (see Extended Data Fig. 2). See a list of full training parameters in Extended Data Fig. 7. We tested each trained model five times around the test track, without input perturbations, and two times while the input was disrupted by a zero-mean Gaussian distribution with variances 0.1, 0.2 and 0.3. Each evaluation consists of driving the car around one cycle of the outermost path of the track, in the anticlockwise direction. We started an evaluation by placing the vehicle at a designated initial location, accelerating the car up to a constant speed of $4.47\,\mathrm{m\,s^{-1}}$ and delegating the steering system's control to the neural network. Every time the vehicle was manoeuvred off the road, we manually steered the car back on track and reported a crash (Fig. 4a).

We connected a random number generator to the input stream to test a model under noise, which added zero-mean Gaussian noise to the camera images. The variance of the Gaussian distribution was determined by the noise intensity levels: 0.1, 0.2 and 0.3. The input images were scaled to the range [0, 1] before the addition of the noise. To induce the same noise pattern for all models, we fixated the initial seed of the random number generator to a constant value.

Models and the training procedure. The architecture of the convolutional layers of each model is given in Extended Data Figs. 5 and 6. Next, we describe the training procedure of the models. If not stated otherwise, this description applies to the passive and active test scenarios. We formulated the end-to-end autonomous driving as a regression task. Hence, we adopted the square error as the training loss function. As recordings of curves and turns are underrepresented in the training data, we multiplied a weighting factor to the loss value of each sample. This weighting factor $w(y) := \exp(|y|\alpha)$ depends on the target curvature y exponentially, thus assigns a higher priority to samples containing road curves and turns. As the test track is located in a forest area where trees cast shadows with variable profiles on the road, we implemented a shadow augmentation data technique during training. In essence, we draw a semi-transparent black or white line over each training image. The location, orientation and width of lines are randomly sampled from uniform distributions. We trained all models, except the feedforward CNN, on subsequences of 16 time steps, which correspond to 0.53 real-time seconds. The neural state-of-standard CT-RNN and LSTM implementations are unbounded, which may lead to instabilities during closed-loop testing, as they are only trained on finite sequences. To avoid the internal states of the controller to grow indefinitely, that is, we apply a clipping operation to the states of the CT-RNN and LSTM to keep the values within the range [−5, 5]. We used Adam[69] as the optimization algorithm with parameters shown in Extended Data Fig. 7. Models' performance and their termination condition is given in Extended Data Fig. 8.

```
Algorithm 1. Training algorithm.
Require: Training set $(X, Y)$, validation set $(\hat{X}, \hat{Y})$, neural network $f(x, w) \mapsto y$,
loss $L(y, \hat{y}) \mapsto \mathbb{R}$, minibatch size $k$
  Initialize weights $w$
  $l_{\text{best}} := \infty$
    for $e = 1 \ldots max_{epochs}$ do
      for $i = 1 \ldots \lfloor |X|/k \rfloor$ do
        $(x, y)$ is random batch of size $k$ from $(X, Y)$
        $w := w - \alpha \frac{\partial L(y, f(x,w))}{\partial w}$ ▽ stochastic gradient descent
      end for
      $l_e := \frac{1}{|\hat{X}|} \sum_{(x,y) \in (\hat{X},\hat{Y})} L(y, f(x,w))$ ▽ validation loss
      if $l_e < l_{\text{best}}$ then
        $l_{\text{best}} := l_e$
        $w_{\text{best}} := w$
      end if
    end for
    return $w_{\text{best}}$
```

```
Algorithm 2. Create NCP architecture.
Require: Set of sensory neurons $N_s$, set of inter-neurons $N_i$, set of command
neuron $N_c$, set of motor neurons $N_m$, density parameters $k_s$, $k_i$, $k_c$ and $k_m$. Allocate
graph $(V, E)$ with $V = N_s \cup N_i \cup N_c \cup N_m$ and $E = \{\}$
  call Algorithm 3 Connect sensory to inter neurons
  call Algorithm 4 Connect intern to command neurons
  call Algorithm 5 Recurrently connect command neurons
  call Algorithm 6 Connect command to motor neurons
  return $(V, E)$
```

The convolutional layers' architecture for all RNN models are listed in Extended Data Fig. 5. After the last convolutional layer, we applied four per-channel linear layers to obtain $8 \times 4 = 32$ latent features serving as sensory inputs to the RNN compartment. We empirically tuned the learning rates and the convolutional layers' hyperparameters and evaluated them on the passive dataset. We observed that NCP took advantage of a lower learning rate for the convolutional layers and a higher learning rate for the RNN compartment. We apply a per-image whitening filter to the images before feeding them into the networks.

NCP design algorithm. The architecture of an NCP is designed by procedurally calling connecting subroutines given by Algorithm 2.

```
Algorithm 3. Subroutine—connect sensory to inter-neurons.
  for all $s \in N_s$ do
    $T$ is random permutation of the set $N_i$
    for $i = 1 \ldots k_s$ do
      $p$ is random variable of distribution $\{50\% \mapsto 1, 50\% \mapsto -1\}$
      Add synapse $(s \to T_i)$ to $E$ with polarity $p$
    end for
  end for
  $\mu_{\text{in}} \leftarrow \frac{1}{|N_i|} \sum_{t \in N_i} |\{s | (s \to t) \in E\}|$ ▽ Compute mean fan-in of neurons $N_i$
  for all $t \in N_i$ such that $\nexists s$: $(s \to t) \in E$ do
    $S$ is random permutation of the set $N_s$
    for $i = 1 \ldots \mu_{\text{in}}$ do
      $p$ is random variable of distribution $\{50\% \mapsto 1, 50\% \mapsto -1\}$
      Add synapse $(S_i \to t)$ to $E$ with polarity $p$
    end for
  end for
```

```
Algorithm 4. Subroutine—connect inter- to command neurons.
  for all $s \in N_i$ do
    $T$ is random permutation of the set $N_c$
    for $i = 1 \ldots k_i$ do
      $p$ is random variable of distribution $\{50\% \mapsto 1, 50\% \mapsto -1\}$
      Add synapse $(s \to T_i)$ to $E$ with polarity $p$
    end for
  end for
  $\mu_{\text{in}} \leftarrow \frac{1}{|N_c|} \sum_{t \in N_c} |\{s | (s \to t) \in E\}|$ ▽ Compute mean fan-in of neurons $N_c$
  for all $t \in N_c$ such that $\nexists s$: $(s \to t) \in E$ do
    $S$ is random permutation of the set $N_i$
    for $i = 1 \ldots \mu_{\text{in}}$ do
      $p$ is random variable of distribution $\{50\% \mapsto 1, 50\% \mapsto -1\}$
      Add synapse $(S_i \to t)$ to $E$ with polarity $p$
    end for
  end for
```

```
Algorithm 5. Subroutine—recurrently connect command neurons.
  for $i = 1 \ldots k_c$ do
    $s$ is random element from $N_c$
    $t$ is random element from $N_c$
    $p$ is a random variable of distribution $\{50\% \mapsto 1, 50\% \mapsto -1\}$
    Add synapse $(s \to t)$ to $E$
    with polarity $p$
  end for
```

```
Algorithm 6. Subroutine—connect command to motor neurons.
  for all $t \in N_m$ do
    $S$ is random permutation of the set $N_c$
    for $i = 1 \ldots k_m$ do
      $p$ is random variable of distribution $\{50\% \mapsto 1, 50\% \mapsto -1\}$
      Add synapse $(S_i \to t)$ to $E$, polarity $p$
    end for
  end for
```

Comparing network sizes. Table 2 illustrates the compactness of the NCP networks compared with the other deep learning counterparts.

Computing saliency maps of convolutional layers. Saliency maps are interpretation methods to visualize the inner workings of a trained neural network by highlighting parts of the input image that contributed most to the decision of a network. We employ saliency maps to analyse what our networks have learned to attend qualitatively. In particular, we are interested in how layers that are common to all tested architectures evolve differently during training. Consequently, we narrow our analysis to the convolutional layers at the beginning of the network. We adopted a technique named VisualBackProp[62] that has been developed deliberately

Table 2 | Network size comparison

Model	Convolutional layers parameters	RNN neurons	RNN synapses	RNN trainable parameters
CNN	5,068,900	-	-	-
CT-RNN	79,420	64	6,112	6,273
LSTM	79,420	64	24,640	24,897
NCP	79,420	**19**	**253**	**1,065**

for autonomous driving research, to compute the saliency maps presented. This method leverages the property of the rectified linear unit (ReLU) activation that the value of each neuron in the feature map is either positive or zero.

```
Algorithm 7. Compute saliency map
  Inputs: Convolutional feature maps $h_1, h_2, \ldots h_N$
  $s :=$ average-over-channel-dimension($h_N$)
  for $i$ from $N-1$ to 1 do
    $z :=$ average-over-channel-dimension ($h_i$)
    $s := z \odot$ deconvolution-to-size-of ($s$,$z$)
  end for
  return $s$.
```

In Algorithm 7, $\odot$ represents the element-wise multiplication, and the deconvolution-to-size-of function scales the first argument to the dimension of the second argument by applying a deconvolution operation.

Structural similarity index. SSIM is a method to compare quality of given images[70]. It is computed as the product of three comparison criteria, the luminance l, contrast c and structure s for given images x and y as follows[70]:

$$\mathrm{SSIM}(x,y) = [l(x,y)]^{\alpha}[c(x,y)]^{\beta}[s(x,y)]^{\gamma}, \tag{10}$$

where:

$$l = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1}, \quad c = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2}, \quad s = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3}. \tag{11}$$

Here, C_1, C_2 and C_3 are the regularization constants, μ_x and μ_y are the means of x and y, σ_x and σ_y are standard deviations and σ_{xy} is the cross-covariance of x and y. In the analysis provided in Fig. 4b, we computed the SSIM for pairs of saliency maps at each time frame (overall 200 frames), between a noise-free version as reference and a perturbed one resulted from input noise injections. We set the exponents $\alpha = \beta = \gamma = 1$, the regularization components $C_1 = (0.01L)^2$, $C_2 = (0.03L)^2$ and $C_3 = C_2/2$, with $L = 255$ corresponding to the dynamics range of the input image values.

Lipschitz continuity computation. The limitation on the speed of change of a function can be computed by the Lipschitz continuity criteria ($|f(t_2) - f(t_1)| \le L|t_2 - t_1|$). The smaller is the Lipschitz constant (L), the smoother the transitions of function f are. The following algorithm computes the maximum Lipschitz constants for neural state activity of RNNs $X(t)$, for an episode of active testing for all RNN types reported in Fig. 4c, we run Algorithm 8.

```
Algorithm 8. Compute maximum Lipschitz constants.
  Inputs: $X^{(N \times T)}$ $N =$ number of neurons, $T =$ length of the test episode
  for $n$ from 1 to $N$ do
    for $t$ from 1 to $T-1$ do
      $L(n,t) = \frac{dX}{dt} \approx X(n,t+1) - X(n,t)/\Delta t, \Delta t = 1$
    end for
  end for
  $L_{max}^{(N\times 1)} = \max\left(L^{(N,T)}\right)$
  Return $L_{\mathrm{max}_{\mathrm{sorted}}} = \mathrm{sort}(L_{\mathrm{max}})$.
```

Principal component analysis. For the analysis provided in Fig. 5f–n, we computed the PCs of the activity of individual neurons in every RNN network, collected from episodes of active driving tests. The output trajectory (steering command) is not included in the PCA analysis, to investigate how the global purpose of the network is expressed by the main PCs of a network's neurons, standalone.

Complementary experiment elaborating on the results of the PCA analysis. Extended Data Fig. 1a represents the dynamics of 64 LSTM cells as a function of the steering control output trajectory. These plots are qualitative representations of the cross-correlation between individual hidden neuron dynamics and the output. A positive slope in these plots denotes a positive correlation, while a negative slope is a sign of a negative correlation. A vertical or horizontal shape is a sign of independence. Extended Data Fig. 1b represents the same space for 19 hidden neurons of an NCP network. LSTM cells behave more arbitrarily and dispersed, while NCP neurons realize a consistent correlation pattern to that of their outputs. This behaviour (conciseness) is well explained by the PCA results shown in Fig. 5. Moreover, we confirmed the concise representation of the NCP network dynamics compared with that of LSTMs by an additional computational experiment. Moving from the PCA results achieved when the network output is included in the experiments, to the setting in which we only consider PCs of the hidden states, we observe that the concise representation for the LSTM network is shattered (Extended Data Fig. 1c,d). This is less apparent in the case of an NCP network, where we observe that the hidden states without having information about the output, can concisely explain the dynamics of driving predominantly by only two PCs (Extended Data Fig. 1e,f).

Data availability

A description of how to obtain the data and code used for this manuscript is available at the manuscript's GitHub repository: https://github.com/mlech26l/keras-ncp/ (https://doi.org/10.5281/zenodo.3999484). The data generated by the active test runs is available for download from the repository, while the full dataset of 193 GB is available on request from M.L.

Code availability

An Apache-2.0 licensed reference implementation maintained by the authors is available at the GitHub repository: https://github.com/mlech26l/keras-ncp/ (https://doi.org/10.5281/zenodo.3999484)

Received: 10 March 2020; Accepted: 10 September 2020;
Published online: 13 October 2020

References

1. Lecun, Y., Cosatto, E., Ben, J., Muller, U. & Flepp, B. *Dave: Autonomous Off-road Vehicle Control Using End-to-end Learning* Technical Report DARPA-IPTO Final Report (Courant Institute/CBLL, 2004); https://cs.nyu.edu/~yann/research/dave/
2. Bojarski, M. et al. End to end learning for self-driving cars. Preprint at http://arXiv.org/abs/1604.07316 (2016).
3. Kato, S. et al. Global brain dynamics embed the motor command sequence of *Caenorhabditis elegans*. *Cell* **163**, 656–669 (2015).
4. Stephens, G. J., Johnson-Kerner, B., Bialek, W. & Ryu, W. S. Dimensionality and dynamics in the behavior of *C. elegans*. *PLoS Comput. Biol.* **4**, e1000028 (2008).
5. Gray, J. M., Hill, J. J. & Bargmann, C. I. A circuit for navigation in caenorhabditis elegans. *Proc. Natl Acad. Sci. USA* **102**, 3184–3191 (2005).
6. Yan, G. et al. Network control principles predict neuron function in the *Caenorhabditis elegans* connectome. *Nature* **550**, 519–523 (2017).
7. Cook, S. J. et al. Whole-animal connectomes of both *Caenorhabditis elegans* sexes. *Nature* **571**, 63–71 (2019).
8. Kaplan, H. S., Thula, O. S., Khoss, N. & Zimmer, M. Nested neuronal dynamics orchestrate a behavioral hierarchy across timescales. *Neuron* **105**(3), 562–576 (2019).
9. LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. *Nature* **521**, 436–444 (2015).
10. Hassabis, D., Kumaran, D., Summerfield, C. & Botvinick, M. Neuroscience-inspired artificial intelligence. *Neuron* **95**, 245–258 (2017).
11. Rudin, C. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nat. Mach. Intell.* **1**, 206–215 (2019).
12. Mnih, V. et al. Human-level control through deep reinforcement learning. *Nature* **518**, 529–533 (2015).
13. Silver, D. et al. Mastering the game of go with deep neural networks and tree search. *Nature* **529**, 484–489 (2016).
14. Silver, D. et al. Mastering the game of go without human knowledge. *Nature* **550**, 354–359 (2017).
15. Schrittwieser, J. et al. Mastering atari, go, chess and shogi by planning with a learned model. Preprint at http://arXiv.org/abs/1911.08265 (2019).
16. Vinyals, O. et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* **575**, 350–354 (2019).
17. Bengio, Y., Courville, A. & Vincent, P. Representation learning: a review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.* **35**, 1798–1828 (2013).
18. Lipton, Z. C. The mythos of model interpretability. *Queue* **16**, 31–57 (2018).
19. Lechner, M., Hasani, R., Rus, D. & Grosu, R. Gershgorin loss stabilizes the recurrent neural network compartment of an end-to-end robot learning scheme. In *Proc. 2020 International Conference on Robotics and Automation (ICRA)* 5446–5452 (2020).
20. Knight, J. C. Safety critical systems: challenges and directions. In *Proc. 24th International Conference on Software Engineering* 547–550 (2002).
21. Pearl, J. *Causality* (Cambridge Univ. Press, 2009).
22. Peters, J., Janzing, D. & Schölkopf, B. *Elements of Causal Inference: Foundations and Learning Algorithms* (MIT Press, 2017).

23. Joseph, M., Kearns, M., Morgenstern, J. H. & Roth, A. Fairness in learning: classic and contextual bandits. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)* 325–333 (2016).
24. Fish, B., Kun, J. & Lelkes, Á. D. A confidence-based approach for balancing fairness and accuracy. In *Proc. SIAM International Conference on Data Mining* 144–152 (2016).
25. Vaswani, A. et al. Attention is all you need. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)* 5998–6008 (2017).
26. Xu, H., Gao, Y., Yu, F. & Darrell, T. End-to-end learning of driving models from large-scale video datasets. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition* 2174–2182 (2017).
27. Amini, A., Paull, L., Balch, T., Karaman, S. & Rus, D. Learning steering bounds for parallel autonomous systems. In *IEEE International Conference on Robotics and Automation (ICRA)* 1–8 (2018).
28. Fridman, L. et al. MIT advanced vehicle technology study: large-scale naturalistic driving study of driver behavior and interaction with automation. *IEEE Access* **7**, 102021–102038 (2019).
29. LeCun, Y. et al. Handwritten digit recognition with a back-propagation network. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)* 396–404 (1990).
30. Amini, A., Rosman, G., Karaman, S. & Rus, D. Variational end-to-end navigation and localization. In *Proc. 2019 International Conference on Robotics and Automation (ICRA)* 8958–8964 (2019).
31. Hochreiter, S. *Untersuchungen zu dynamischen neuronalen netzen*. Diploma, Technische Universität München 91 (1991).
32. Bengio, Y., Simard, P. & Frasconi, P. et al. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Netw.* **5**, 157–166 (1994).
33. Rumelhart, D. E., Hinton, G. E. & Williams, R. J. Learning representations by back-propagating errors. *Nature* **323**, 533–536 (1986).
34. Hochreiter, S. & Schmidhuber, J. Long short-term memory. *Neural Comput.* **9**, 1735–1780 (1997).
35. Reimer, B., Mehler, B., Wang, Y. & Coughlin, J. F. A field study on the impact of variations in short-term memory demands on drivers' visual attention and driving performance across three age groups. *Hum. Factors* **54**, 454–468 (2012).
36. Funahashi, K.-i & Nakamura, Y. Approximation of dynamical systems by continuous time recurrent neural networks. *Neural Netw.* **6**, 801–806 (1993).
37. Chen, T. Q., Rubanova, Y., Bettencourt, J. & Duvenaud, D. K. Neural ordinary differential equations. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)* 6571–6583 (2018).
38. Lechner, M. & Hasani, R. Learning long-term dependencies in irregularly-sampled time series. Preprint at http://arXiv.org/abs/2006.04418 (2020).
39. Sarma, G. P. et al. Openworm: overview and recent advances in integrative biological simulation of *Caenorhabditis elegans*. *Phil. Trans. R. Soc. B* **373**, 20170382 (2018).
40. Gleeson, P., Lung, D., Grosu, R., Hasani, R. & Larson, S. D. c302: a multiscale framework for modelling the nervous system of *Caenorhabditis elegans*. *Phil. Trans. R. Soc. B.* **373**, 20170379 (2018).
41. Hasani, R., Lechner, M., Amini, A., Rus, D. & Grosu, R. Liquid time-constant networks. Preprint at http://arXiv.org/abs/2006.04439 (2020).
42. LeCun, Y. et al. Backpropagation applied to handwritten zip code recognition. *Neural Comput.* **1**, 541–551 (1989).
43. Wicks, S. R., Roehrig, C. J. & Rankin, C. H. A dynamic network simulation of the nematode tap withdrawal circuit: predictions concerning synaptic function using behavioral criteria. *J. Neurosci.* **16**, 4017–4031 (1996).
44. Lechner, M., Hasani, R., Zimmer, M., Henzinger, T. A. & Grosu, R. Designing worm-inspired neural networks for interpretable robotic control. In *International Conference on Robotics and Automation (ICRA)* 87–94 (2019).
45. Hasani, R., Lechner, M., Amini, A., Rus, D. & Grosu, R. The natural lottery ticket winner: reinforcement learning with ordinary neural circuits. In *Proc. International Conference on Machine Learning* (2020).
46. Bengio, Y. & Grandvalet, Y. No unbiased estimator of the variance of *k*-fold cross-validation. *J. Mach. Learn. Res.* **5**, 1089–1105 (2004).
47. Molnar, C. *Interpretable Machine Learning* (Lulu.com, 2019).
48. Hasani, R. *Interpretable Recurrent Neural Networks in Continuous-time Control Environments*. PhD dissertation, Technische Universität Wien (2020).
49. Erhan, D., Bengio, Y., Courville, A. & Vincent, P. *Visualizing Higher-layer Features of a Deep Network* Technical Report 1341 (Univ. Montreal, 2009).
50. Zeiler, M. D. & Fergus, R. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision* 818–833 (2014).
51. Yosinski, J., Clune, J., Nguyen, A., Fuchs, T. & Lipson, H. Understanding neural networks through deep visualization. Preprint at http://arXiv.org/abs/1506.06579 (2015).
52. Karpathy, A., Johnson, J. & Fei-Fei, L. Visualizing and understanding recurrent networks. Preprint at http://arXiv.org/abs/1506.02078 (2015).
53. Strobelt, H., Gehrmann, S., Pfister, H. & Rush, A. M. LSTMVis: a tool for visual analysis of hidden state dynamics in recurrent neural networks. *IEEE Trans. Vis. Comput Graph.* **24**, 667–676 (2018).
54. Bilal, A., Jourabloo, A., Ye, M., Liu, X. & Ren, L. Do convolutional neural networks learn class hierarchy? *IEEE Trans. Vis. Comput. Graph.* **24**, 152–162 (2018).
55. Olah, C. et al. The building blocks of interpretability. *Distill* **3**, e10 (2018).
56. Simonyan, K., Vedaldi, A. & Zisserman, A. Deep inside convolutional networks: visualising image classification models and saliency maps. Preprint at http://arXiv.org/abs/1312.6034 (2013).
57. Fong, R. C. & Vedaldi, A. Interpretable explanations of black boxes by meaningful perturbation. *Proc. IEEE International Conference on Computer Vision* 3449–3457 (IEEE, 2017).
58. Kindermans, P.-J., Schütt, K. T., Alber, M., Müller, K.-R. & Dähne, S. Learning how to explain neural networks: PatternNet and PatternAttribution. *Proc. International Conference on Learning Representations (ICLR)* (2018).
59. Sundararajan, M., Taly, A. & Yan, Q. Axiomatic attribution for deep networks. *Proc. 34th International Conference on Machine Learning (ICML)* (2017).
60. Doshi-Velez, F. & Kim, B. Towards a rigorous science of interpretable machine learning. Preprint at http://arXiv.org/abs/1702.08608 (2017).
61. Trask, A. et al. Neural arithmetic logic units. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)* 8035–8044 (2018).
62. Bojarski, M. et al. Visualbackprop: efficient visualization of cnns for autonomous driving. In *IEEE International Conference on Robotics and Automation (ICRA)* 1–8 (2018).
63. Maaten, Lvd & Hinton, G. Visualizing data using t-sne. *J. Mach. Learn. Res.* **9**, 2579–2605 (2008).
64. Tesla Autopilot (Tesla, 2020); https://www.tesla.com/autopilot
65. Karpathy, A. PyTorch at Tesla. In *PyTorch Devcon Conference 19* https://youtu.be/oBklltKXtDE (2019).
66. Press, W. H., Teukolsky, S. A., Vetterling, W. T. & Flannery, B. P. *Numerical Recipes: The Art of Scientific Computing* 3rd edn (Cambridge Univ. Press, 2007).
67. Naser, F. et al. A parallel autonomy research platform. In *2017 IEEE Intelligent Vehicles Symposium (IV)* 933–940 (IEEE, 2017).
68. Amini, A. et al. Learning robust control policies for end-to-end autonomous driving from data-driven simulation. *IEEE Robot. Autom. Lett.* **5**, 1143–1150 (2020).
69. Kingma, D. P. & Ba, J. Adam: a method for stochastic optimization. In *Proc. 3rd International Conference for Learning Representations (ICLR)* (2015).
70. Wang, Z., Bovik, A. C., Sheikh, H. R. & Simoncelli, E. P. et al. Image quality assessment: from error visibility to structural similarity. *IEEE Trans. Image Process.* **13**, 600–612 (2004).
71. Girosi, F., Jones, M. & Poggio, T. Regularization theory and neural networks architectures. *Neural Comput.* **7**, 219–269 (1995).
72. Smale, S. & Zhou, D.-X. Learning theory estimates via integral operators and their approximations. *Constr. Approx.* **26**, 153–172 (2007).

Acknowledgements

We thank M. Zimmer and the Zimmer Group for constructive discussions. R.H. and R.G. are partially supported by Horizon-2020 ECSEL Project grant no. 783163 (iDev40), and the Austrian Research Promotion Agency (FFG), project no. 860424. M.L. and T.A.H. were supported in part by the Austrian Science Fund (FWF) under grant Z211-N23 (Wittgenstein Award). A.A. is supported by the National Science Foundation (NSF) Graduate Research Fellowship Program. A.A. and D.R. were partially sponsored by the United States Air Force Research Laboratory and was accomplished under Cooperative Agreement no. FA8750-19-2-1000. R.H. and D.R. are partially supported by The Boeing Company. This research work is drawn from the PhD dissertation of R.H.

Author contributions

R.H. and M.L. conceptualized, designed and performed research, and analysed data. A.A. contributed to data curation, research implementation and new analytical tools, and analysed data. R.G., T.A.H. and D.R. helped with the design and supervised the work. All authors wrote the paper.

Competing interests

The authors declare no competing interests.

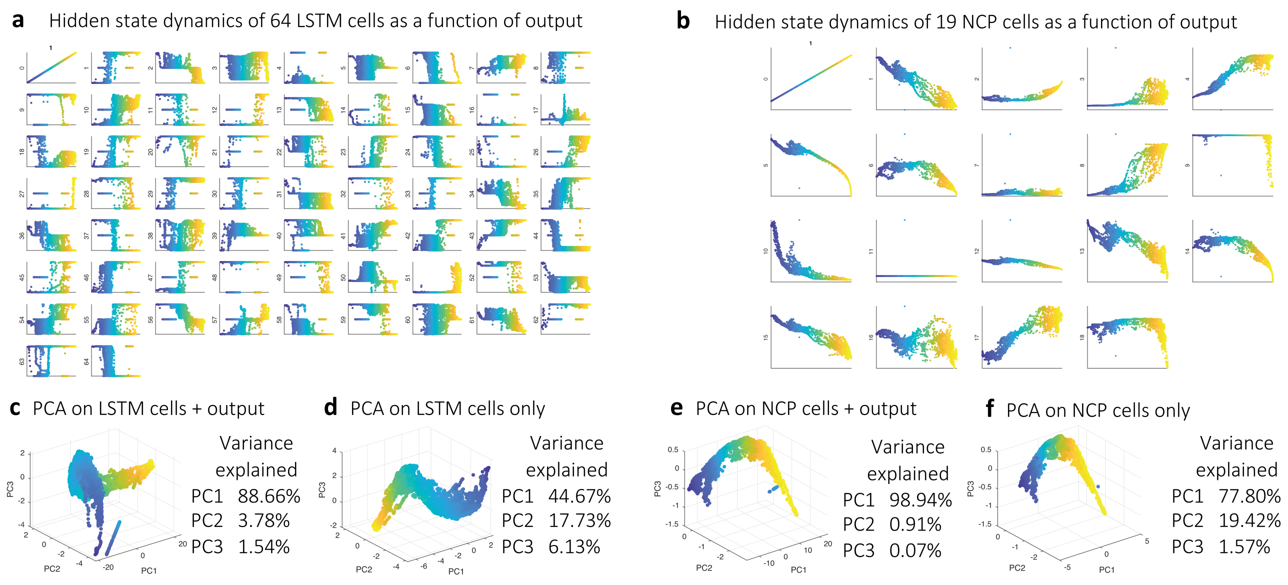
Additional information

Extended data is available for this paper at https://doi.org/10.1038/s42256-020-00237-3.

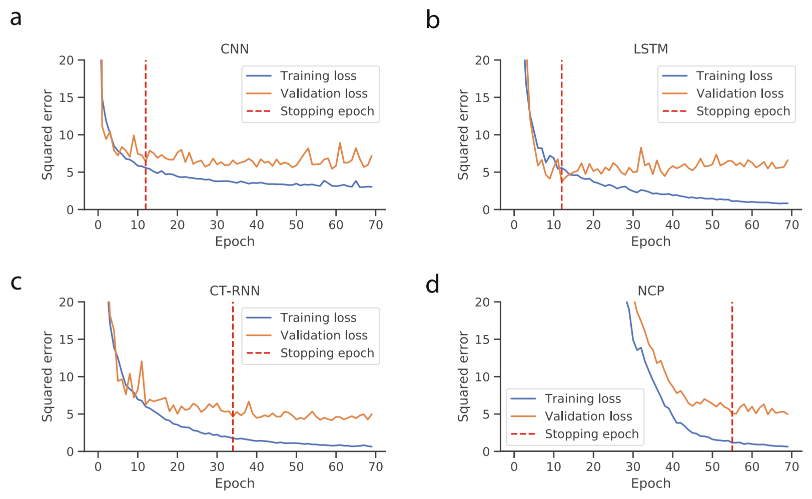
Supplementary information is available for this paper at https://doi.org/10.1038/s42256-020-00237-3.

Correspondence and requests for materials should be addressed to M.L. or R.H.

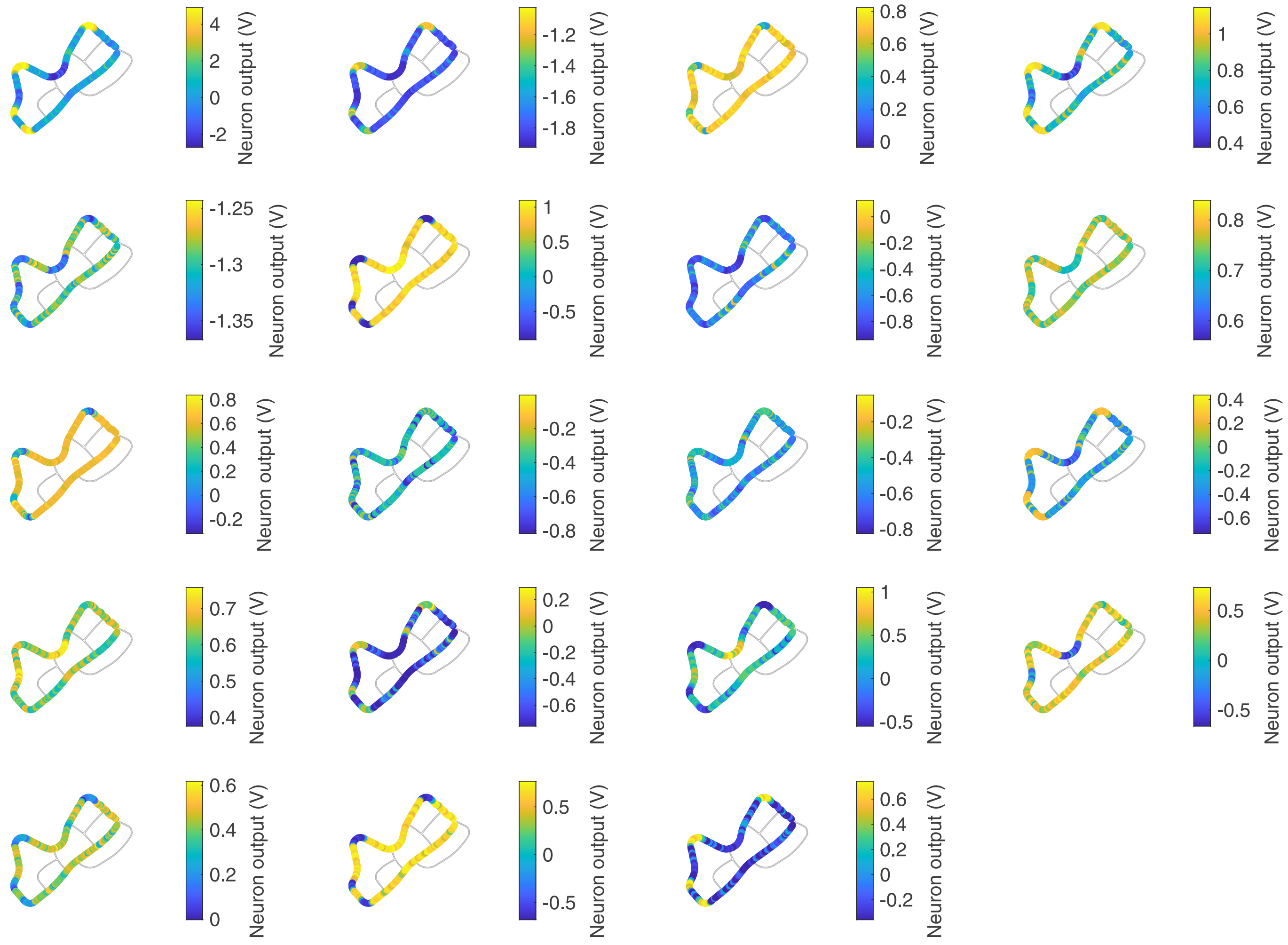
Reprints and permissions information is available at www.nature.com/reprints.



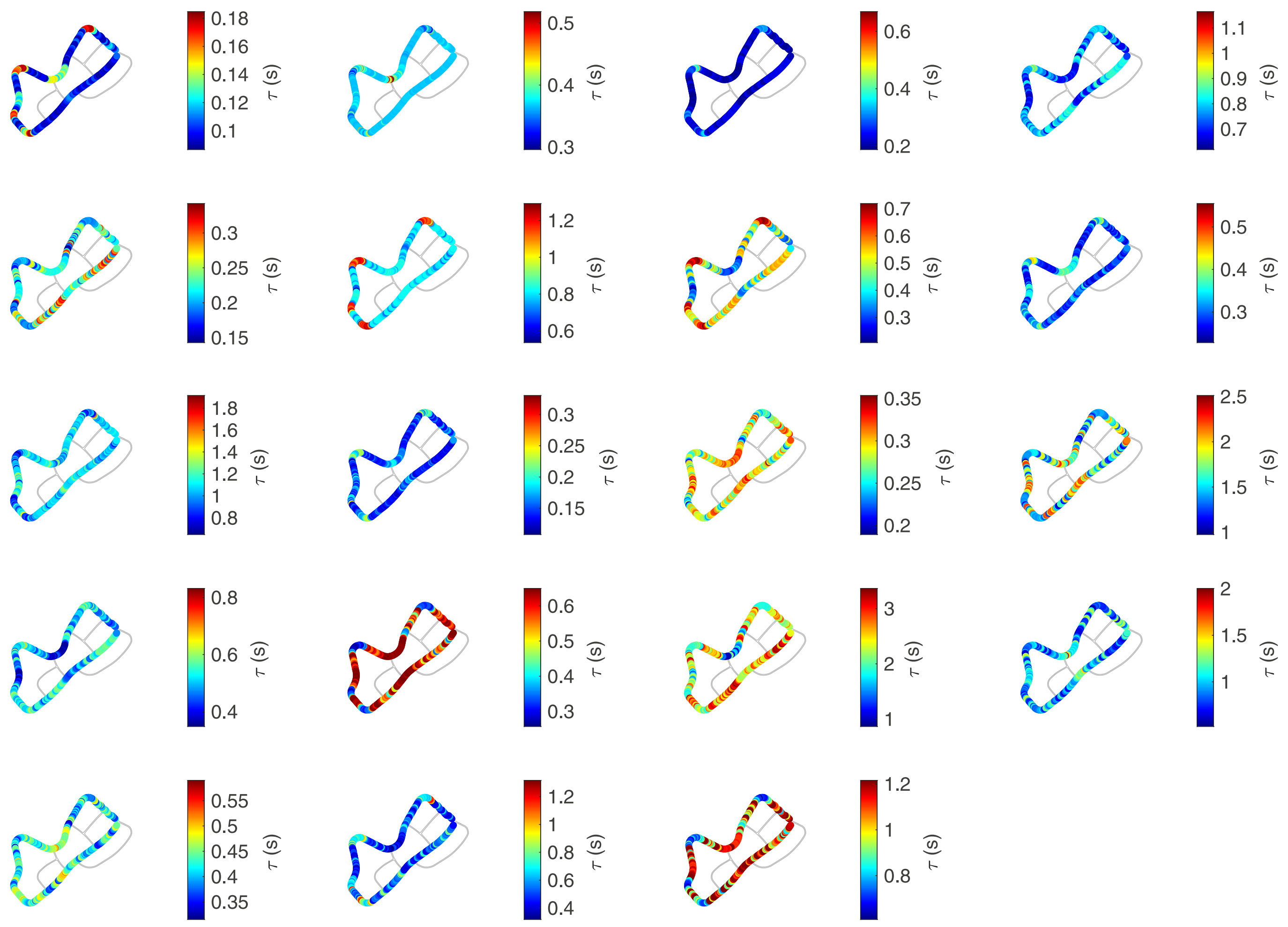
Extended Data Fig. 1 | Conciseness of the hidden-state dynamics of LSTMs vs NCPs. a, Hidden state dynamics of 64 LSTM cells as a function of network output. **b**, Hidden state dynamics of 13 NCP cells as a function of the network output. **c**, PCA on LSTM cells + output, **d**, PCA on LSTM cells only. **e**, PCA on NCP cells + output. **f**, PCA on NCP cells only. x-axis represents the activity of the output, y-axis stands for the dynamics of an individual neuron. The colour represents the steering angle (The more yellow regions depict sharper turns to the left-hand-side, and the more blue regions stand for sharper turns to the right-hand-side).



Extended Data Fig. 2 | Learning curves of the models tested in the active driving experiments. Early stopping [71,72] is deployed as a regularisation mechanism to obtain better generalisation. The terminating epoch for each experiment, is reported in Extended Data Figure 4.



Extended Data Fig. 3 | Neural activity of all NCP neurons presented in Fig. 6. The colour-bar represents the neuron output of each individual neuron in the NCP architecture.



Extended Data Fig. 4 | Coupling sensitivity of all NCP neurons presented in Fig. 6. The colour-bar represents the time constant of each individual neuron in the NCP architecture.

Layer	Filters	Kernel size	Strides
1	24	5	2
2	36	5	2
3	48	3	2
4	64	3	1
5	8	3	1

Extended Data Fig. 5 | Convolutional head. Size of the convolutional kernels.

Layer	Type	Hyperparameters
1	Conv2D	F: 24, K: 5, S: 2
2	Conv2D	F: 36, K: 5, S: 2
3	Conv2D	F: 48, K: 3, S: 2
4	Conv2D	F: 64, K: 3, S: 1
5	Conv2D	F: 64, K: 3, S: 1
6	Flatten	
7	Dropout	δ_1
8	Fully-connected	U: 1000, ReLU
9	Dropout	δ_2
10	Fully-connected	U: 100, ReLU
11	Dropout	δ_3
12	Fully-connected	U: 1, Identity

Extended Data Fig. 6 | Layers of the feedforward CNN, adapted from [2]. Conv2D refers to a convolutional layer, F to the number of filters, K to the kernel size, S to the strides, U to the number of units in a fully-connected layer. The values of the dropout-rates δ_1,δ_2, and δ_3 were optimised on the passive benchmark and reported in Extended Data Figure 3.

Variable	Value	Comment
Input resolution	200-by-78	Pixels
Input channels	3	8-bit RGB colour space
Learning rate	$5 \cdot 10^{-4}$	CNN, LSTM, CT-RNN
Learning-rate	$1 \cdot 10^{-3}$	NCP (RNN compartment)
Learning-rate	$2.5 \cdot 10^{-5}$	NCP (convolutional head)
β_1	0.9	Parameter of Adam
β_2	0.999	Parameter of Adam
Minibatch-size	20	
Training sequences length	16	time-steps
α	0.1	Parameter in the weighting factor
Max. number of training epochs	100	Validation set determines actual epochs
δ_1	0.5	Dropout-rate of the CNN
δ_2	0.5	Dropout-rate of the CNN
δ_3	0.3	Dropout-rate of the CNN

Extended Data Fig. 7 | Models' training hyperparameters. The values of all hyperparameters were selected through empirical evaluation over the passive training dataset. We did not search through the hyperparameters space exhaustively, due to the computational costs. However, the use of a systematic meta-learning algorithm over these parameter-spaces can presumably result in achieving better performances.

Model	Stopping epoch	Training loss	Validation loss
CNN	12	5.58	6.44
CT -RNN	34	1.77	4.62
LSTM	12	5.58	3.70
NCP	55	1.15	5.09

Extended Data Fig. 8 | The learning termination epoch properties, (shown in Extended Data Fig. 2). Training and validation metrics of the models tested in the active driving experiment. As discussed thoroughly (Fig. 4), LSTM model achieves the best performance in the passive test but fails to express proper driving behaviour under environmental disturbances.