A deep-learning approach to realizing functionality in nanoelectronic devices

ARTICLES
https://doi.org/10.1038/s41565-020-00779-y
nature nanotechnology

A deep-learning approach to realizing functionality in nanoelectronic devices

Hans-Christian Ruiz Euler[1,4], Marcus N. Boon[1,4], Jochem T. Wildeboer[1], Bram van de Ven[1], Tao Chen[1], Hajo Broersma[2], Peter A. Bobbert[1,3] and Wilfred G. van der Wiel[1]✉

Many nanoscale devices require precise optimization to function. Tuning them to the desired operation regime becomes increasingly difficult and time-consuming when the number of terminals and couplings grows. Imperfections and device-to-device variations hinder optimization that uses physics-based models. Deep neural networks (DNNs) can model various complex physical phenomena but, so far, are mainly used as predictive tools. Here, we propose a generic deep-learning approach to efficiently optimize complex, multi-terminal nanoelectronic devices for desired functionality. We demonstrate our approach for realizing functionality in a disordered network of dopant atoms in silicon. We model the input–output characteristics of the device with a DNN, and subsequently optimize control parameters in the DNN model through gradient descent to realize various classification tasks. When the corresponding control settings are applied to the physical device, the resulting functionality is as predicted by the DNN model. We expect our approach to contribute to fast, in situ optimization of complex (quantum) nanoelectronic devices.

Exploring the behaviour of nanoelectronic devices can be a time-consuming task that requires considerable human and experimental resources. For instance, multi-terminal nanoelectronic devices for quantum technology[1–5] and hardware-based computational paradigms[6–10] require delicate tuning of control voltages to achieve a desired functionality. Manual tuning becomes increasingly challenging for devices with a growing parameter space. As the number and complexity of interconnected nanoelectronic devices increase, the demand for automated tuning methods rises as well. Existing methods rely either on search heuristics[1,3,4] or, increasingly, on machine-learning methods that combine measurements with either image analysis[11–15] or gradient estimation of the output response[5] to converge iteratively to the desired functionality.

This article proposes the use of a deep neural network (DNN)[16] model of a nanoelectronic device for optimizing the values of the control settings of the device to achieve a desired functionality. Tuning the corresponding control parameters of the DNN model (instead of the control settings of the physical device) has several advantages. These include a substantial reduction in tuning time, as well as in human and experimental resources, because the control parameters can be tuned in a completely automated manner with minimal physical measurements. The model is created by training a DNN with a measured training data set that represents the input–output characteristics of a multi-terminal nanoelectronic device. DNNs have been shown to act as efficient function approximators[17] of multidimensional functions, and can be trained by using an optimization algorithm known as stochastic gradient descent[18–20]. Until now, in physics, DNNs have been introduced mainly as a predictive tool[21–25]. Here, we demonstrate that we can realize functionality successfully in a nanoelectronic device by optimizing its DNN model, which acts as a proxy of the physical device, through gradient descent in a fast and fully automated way.

Here, we illustrate our general approach of creating a DNN model of a multi-terminal nanoelectronic device, and the process for obtaining the desired functionality (Fig. 1). For the case study of this article, voltages are applied at the input terminals and currents are measured at the output terminals. First, we sample the multidimensional space of input voltages to obtain a sufficiently large amount of input–output data. Next, we set up a DNN architecture with the numbers of inputs and outputs matching those of the device. The DNN should have a sufficiently large number of hidden layers and neural nodes per layer to describe accurately the input–output data (see Methods and Supplementary Fig. 1). Then, the DNN is trained with measured training data and tested with unseen measured data. The desired functionality is realized as follows. We assign some input terminals as 'control' terminals (Fig. 1, circles). The control voltages applied at these terminals control the input–output characteristics between the output currents and the voltages applied at the input terminals (squares). At this stage, the control voltages that are to be applied to the device become the learnable control parameters of the DNN. A desired functionality, defined as a specific targeted input–output relationship, is then searched for in the DNN model by using gradient descent on the control parameters. Finally, the obtained functionality is verified by applying the obtained corresponding control voltages directly to the physical device, without any further experimental optimization.

We demonstrate this approach for a multi-terminal nanoelectronic device that we investigated recently[6]. The device consists of an electrically tuneable network of boron dopants in silicon (Si:B) with eight terminals (electrodes), seven of which act as voltage inputs and one as current output. The active area of the device has a diameter of about 300 nm. The device can be tuned to solve problems of two-dimensional categorical nonlinear binary classification[26] by using an evolutionary approach[27]. Boolean functionality is realized

[1]NanoElectronics Group, MESA+ Institute for Nanotechnology, and Center for Brain-Inspired Nano Systems (BRAINS), University of Twente, Enschede, The Netherlands. [2]Programmable Nanosystems and Formal Methods and Tools, MESA+ Institute for Nanotechnology, DSI Digital Society Institute, and Center for Brain-Inspired Nano Systems (BRAINS), University of Twente, Enschede, The Netherlands. [3]Molecular Materials and Nanosystems and Center for Computational Energy Research, Department of Applied Physics, Eindhoven University of Technology, Eindhoven, The Netherlands. [4]These authors contributed equally: Hans-Christian Ruiz Euler, Marcus N. Boon. ✉e-mail: W.G.vanderWiel@utwente.nl

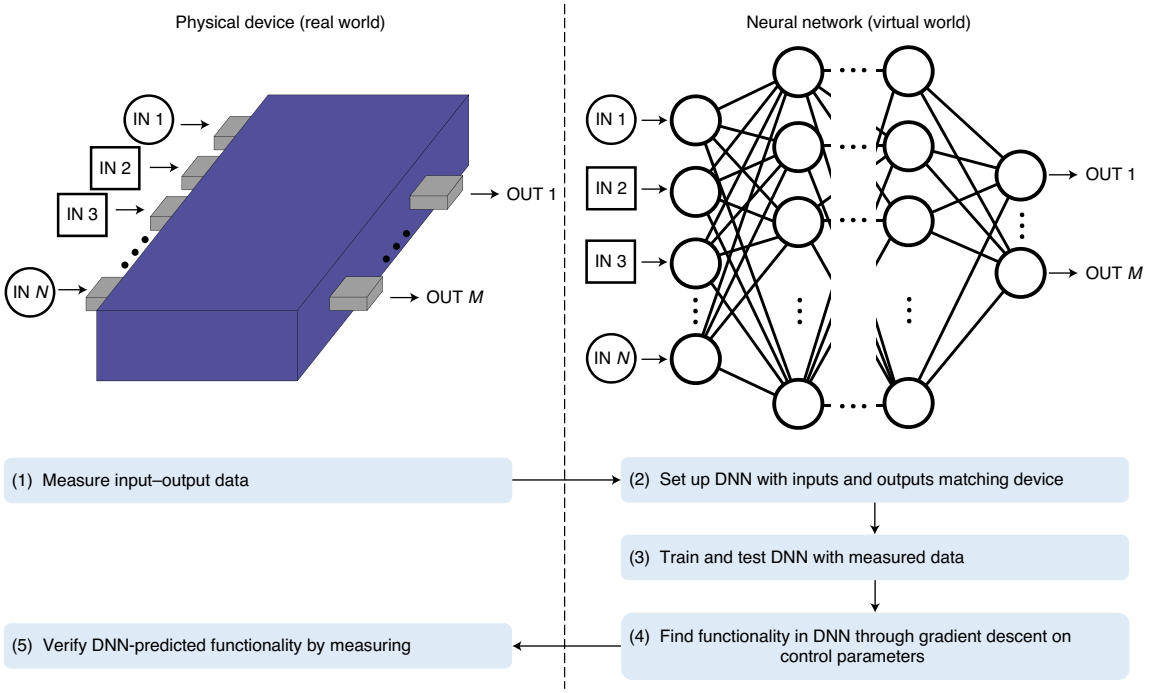

Fig. 1 | Realizing functionality in a nanoelectronic device by using a DNN model. The following steps are adopted to create a DNN model that captures the input–output characteristics of a multi-terminal nanoelectronic device with *N* input and *M* output terminals, and to realize a desired functionality. First, input–output data of the device are measured (1). Next, a sufficiently deep and wide DNN is set up, with numbers of inputs and outputs matching those of the device (2). The DNN is trained with the measured training data and tested with unseen test data (3). The DNN is used to find the desired functionality through gradient descent on the control parameters (4). The predicted corresponding control voltages are then applied to the physical device to verify the functionality (5). Circles and squares pointing towards the device or DNN indicate the control and data inputs, respectively. Circles in the DNN represent artificial neurons and their activation function.

by applying four input voltage combinations to two input terminals, which correspond to data inputs 00, 01, 10 and 11, and by tuning the remaining terminal voltages such that the four data combinations are mapped to the desired logic gate that is represented by its output current levels. Typical I–V characteristics measured at 77 K (Fig. 2a) demonstrate the nonlinear dependence of the output current I_{out} at terminal 8 as a function of the voltage applied at each of the other seven terminals while the remaining terminals are grounded.

Deep neural network modelling

To train a DNN model of the device, the seven-dimensional space of its input voltages must be sampled and the corresponding output current measured. One way to obtain these input–output data would be to sample the input voltage space uniformly and randomly, which would be optimal when no information on the input–output relation is available. However, the associated abrupt voltage jumps cause a transient response, namely a time-dependent current that is related to capacitive effects. To circumvent this, pauses of at least 20 ms (about five times the RC time constant of the device in combination with the measurement circuit) after input voltage steps could be incorporated to let the system settle into a steady state. As the number of voltage combinations grows exponentially with the number of terminals, this rapidly becomes very time-consuming.

To solve this problem, we sample the space of input voltages by using sinusoidal or triangular modulation functions and a sampling frequency of 50 Hz (Fig. 2b and Methods). This sampling technique minimizes discontinuities in the applied voltages and therefore reduces transient effects. The efficiency of sampling this way depends strongly on the choice of the modulation frequencies. By choosing the modulation frequencies of the different input voltages such that the ratios of all of the frequency combinations are irrational (Supplementary Table 1), we guarantee that the input space is densely covered and that there are no recurrences of voltage combinations. Also, the sampling density can then be increased by simply increasing the sampling time. Our approach has substantial advantages over standard sampling with a predefined uniform grid in the seven-dimensional voltage space. For the same total number of samples and highest modulation frequency as in our approach, which is limited by remaining transient effects, the total sampling time in the standard approach would need to be around five times longer (Supplementary Table 2). In addition, an increase in the sampling density would require performing a new sampling over a new grid instead of simply increasing the sampling time.

The input data consist of tuples of input voltages (V_1 to V_7, in V) and the output data are scalars representing the output currents (I_{out}, in nA). The input and output layers of the DNN correspond to the input terminals and output terminal of the device, respectively. We model our nanoelectronic device with a fully connected DNN consisting of seven inputs and one output. A network architecture with five hidden layers and 90 nodes per layer is found to have a sufficiently small test error (Supplementary Fig. 1). Minimization of the mean squared error is used in training the DNN (see Methods).

When the output current predicted by the DNN is plotted against the measured output current for unseen measured test data (Fig. 2c), only small deviations from the identity curve are observed compared to the overall range of the output. The root mean squared error (RMSE) in the predicted currents for the test data is 1.2 nA (Fig. 2d), which is 0.27% of the total current output range. These results show that the trained DNN predicts the unseen data accurately.

Automatic functionality search through gradient descent

A desired functionality is specified by a targeted dependence of the output current on one or more input voltages of selected terminals. The functionality is obtained by learning the values of the remaining

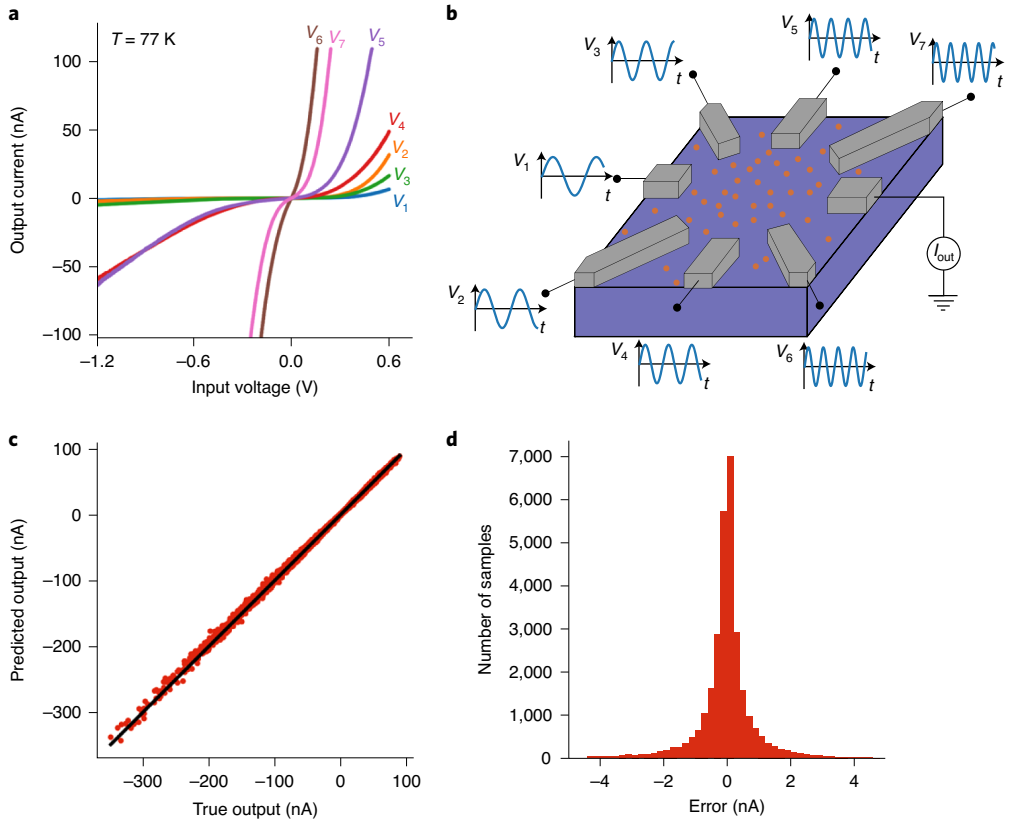

Fig. 2 | Sampling input–output data to train and test the DNN. a, Typical *I*–*V* characteristics of the Si:B device measured at 77 K, showing nonlinear behaviour. The current between the output terminal and the ground is measured while applying a voltage to each of the input terminals and grounding the remaining terminals. **b**, Schematic representation of the device, with boron dopants represented by orange dots (see ref. [6] for details). Training and test data are obtained by measuring the output current (I_{out}) while applying sinusoidal (as shown in this example) or triangular voltage modulations (V_1 to V_7) as a function of time *t* to the input terminals (see Supplementary Table 1 for frequencies). **c**, Output current predicted by the trained DNN for the unseen test data set against the current measured in the physical device. The solid line has a slope of unity. **d**, Histogram of the test error, showing an RMSE of 1.2 nA.

input voltages, that is the control parameters, by following the negative gradient of a cost function $E(y,z)$, which is a measure of the similarity between the predicted outcome data y and the targeted outcome data z. At this stage, the internal weights of the DNN are kept frozen. We use a cost function composed of a correlation factor and the logistic function (see Methods). In contrast to the mean squared error, this cost function does not target specific output current values, but rather promotes separation of low ('0') and high ('1') output currents. The entire process of automated optimization by the DNN model is represented here for the case of an XOR Boolean logic gate (Fig. 3). To demonstrate the speed and accuracy of this approach, we apply it to solve different tasks that have increasing accuracy requirements.

Classification with DNN-optimized nanoelectronic devices

Starting with Boolean logic gates, voltages V_2 and V_3 on terminals 2 and 3 are used as data inputs (Fig. 3a, squares; Fig. 3b, coloured terminals). Hence, there are five remaining control voltages (V_1, V_4, V_5, V_6 and V_7) to realize the gates. We note that it is to a large extent a matter of choice which terminals are used for data input and which are used for control. Optimal functionality is generally obtained when the input terminals are not neighbouring the output terminal or each other. This is intuitively understandable from the underlying physics of variable-range hopping of charge carriers, in between the dopants and between the dopants and the electrodes, in the electrostatic potential generated by the electrode voltages and the dopants and charge carriers themselves[6]. During verification of the predicted gates, the input data (V_2 and V_3) are presented to the physical device as a time sequence in the order shown in Fig. 3b. These input wave forms, given by the 0011 (olive green) and 0101 (blue) signals, represent all four possible combinations of inputs for the truth table for Boolean logic (00, 01, 10 and 11). We show the numerical prediction and experimental verification of an XOR gate (Fig. 3c, black and red, respectively).

Results are provided for all logic gates AND, NAND, OR, NOR, XOR and XNOR (Fig. 4). The voltage values that correspond to logic inputs '0' and '1' are −1.2 V and 0.6 V, respectively (Fig. 4a). We show the output currents for the logic gates that are predicted by the DNN (Fig. 4b, black). To verify the predicted output currents, the predicted control voltages and the binary input voltages are applied to the physical device, and we obtain the measured output currents (Fig. 4b, red). A comparison shows that all of the gates predicted by the DNN are also demonstrated in the physical device. Moreover, the values of the output currents are predicted with high accuracy. The RMSEs of the predicted gates (values in Fig. 4b) are consistent with the test error of 1.2 nA in Fig. 2d. The normalized RMSEs display the magnitude of the error with respect to the current scale (highest minus lowest current) of the predicted gates. The normalized RMSEs show that the most poorly predicted gates (NAND and OR) have a relative error of 5.6%. Possible threshold current values separating 'true' and 'false' for each logic gate are represented (Fig. 4b, dashed lines). Optimal thresholds could be determined automatically by training a single perceptron[28] on the output and target data.

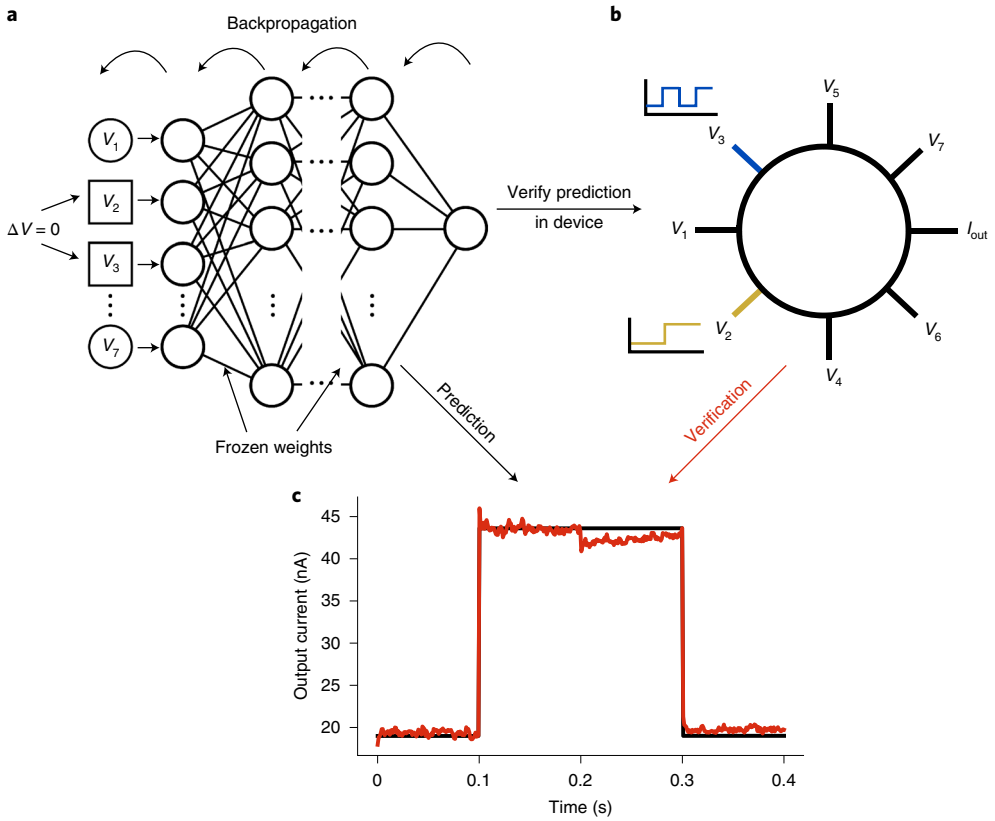

Fig. 3 | DNN prediction of device functionality and verification. a, Backpropagation of the gradient through the DNN to tune the control parameters to realize the desired functionality. During the tuning process, the internal weights of the DNN are frozen. The input and control terminals are represented by squares and circles, respectively. **b**, For this example, the prediction of an XOR Boolean logic gate is chosen, which consists of two-dimensional input data (V_2 and V_3, −1.2 V for '0' and 0.6 V for '1') with the five remaining voltages as control parameters. The control voltages are tuned such that the input data are mapped to the desired outputs. **c**, To verify the predicted outcome, the tuned control voltages and the time-dependent input voltages are applied to the physical device. Each input combination is applied for 0.1 s, which results in a total output signal of 0.4 s; this is sufficiently long to reveal typical fluctuations in the output current. The predicted outcome is shown in black and the physical measurement in red. Between the different input combinations, the input voltages are linearly ramped over 10 ms periods to their new values, during which the current is masked. We note that optimization of the devices and the interfacing equipment may eventually lead to a readout bandwidth exceeding 100 MHz[6].

The DNN model allows for an accelerated exploration of functionality without having to perform further measurements on the device after collecting the training and test data. In our case, we are able to reduce the search of Boolean functionality from 15 minutes per gate by performing physical measurements in combination with an evolutionary algorithm[6], to around 10 seconds on a standard computer (Intel Core i5-8250U CPU processor @ 1.60 GHz with four cores and 8 GB RAM) by using the DNN model, which is an increase in speed of almost two orders of magnitude.

Next, we study a more challenging binary classification problem that consists of classifying points in a two-dimensional feature space with two classes: data points in an outer ring that correspond to the label '0', and data points in an inner cluster that correspond to the label '1' (Fig. 5a). The two classes in this ring classification problem must be separated by using a closed decision boundary. Using in this case V_4 and V_5 from Fig. 3b as data inputs, the DNN is trained to separate the two classes such that the inner class corresponds to a high output current and the outer class to a low output current. The remaining control parameters (V_1, V_2, V_3, V_6 and V_7) are to be tuned. In addition, we have added a scaling factor and two bias parameters to the input voltages, which enable the DNN to determine an affine transformation of the data by itself (see Methods). In the comparison of the resulting prediction by the DNN and the physical measurements (Fig. 5b), the RMSE is 0.78 nA and the normalized RMSE is 4.4%, which demonstrates the accuracy of the prediction by the DNN model. We note that the second input voltage ranges up to 1.1 V, whereas the DNN is only trained for voltages up to 0.6 V for this terminal. The trained DNN therefore successfully extrapolates over a 0.5 V range. To visualize the results, a heat map created by sweeping the two input voltages in the physical device is included (Fig. 5a). This demonstrates that the output current is indeed maximized for the inner class and minimized for the outer class. The two classes are separable by defining a threshold current as decision boundary (Fig. 5b, dashed line). We note that the data inputs V_4 and V_5 are different from those for the Boolean gates (V_2 and V_3). A good functionality can also be obtained with other choices for the data inputs. Another good ring classification functionality is demonstrated by using V_6 and V_7 as data inputs (Supplementary Fig. 5 and Supplementary Tables 3, 4).

Feature mapping with DNN-optimized nanoelectronic devices

We fabricated another device with the same structure as the device used for the previous experiments (Figs. 2b,3b), and used it to perform a more complex task. The task involves distinguishing 16 different 2×2 pixel features by the output current of the device, which is a possible subtask of a higher-level image recognition task[6]. Training and test data are measured as before, except for the use of triangular instead of sinusoidal modulation functions and the use of other voltage amplitudes (Supplementary Table 1). The modelling

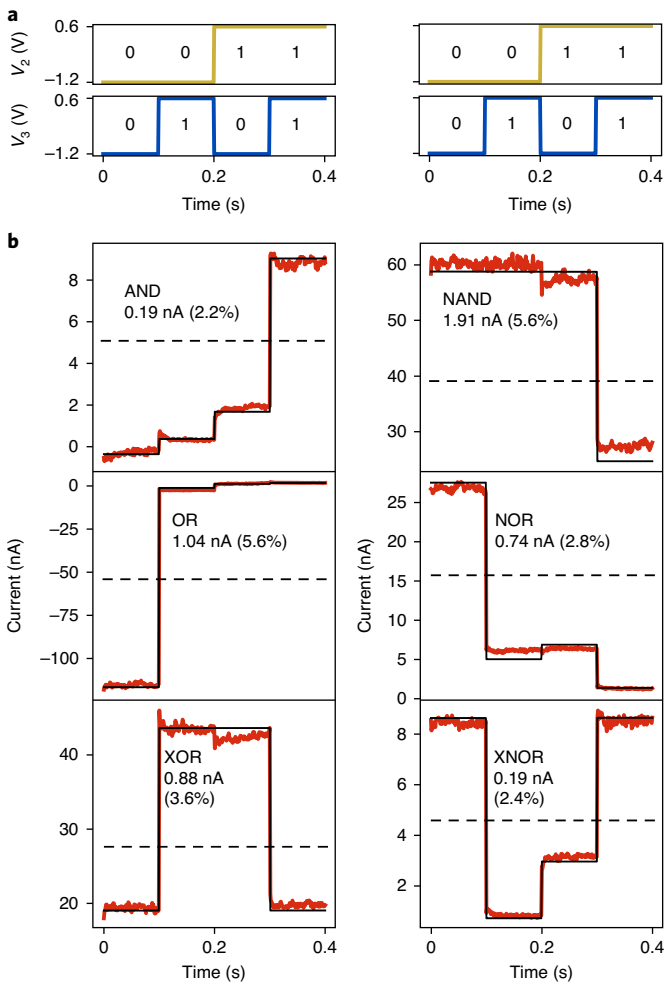

Fig. 4 | Prediction and verification of Boolean logic. a, Applied voltages V_2 and V_3 to test the Boolean logic gates. **b**, Comparison of output currents. Black curves, logic gates predicted by the trained DNN. Red curves, output current measured in the device by using the control voltages predicted by the DNN (given in Supplementary Table 5). Between the different inputs, the voltages are linearly ramped over 10 ms periods to their new values, during which the current is masked. The values in the panels indicate the RMSE in nA and the normalized RMSE (in brackets). The dashed lines indicate current levels that could be used for separating the logical outputs '0' and '1'.

of the DNN is the same as before. The RMSE of the predicted test data happens to have the same value of 1.2 nA as before, which is now 0.6% of the output current range. The features are defined by 2×2 black and white pixel combinations that are converted to high (black) and low (white) voltage values on four input terminals, in this case V_2, V_3, V_4 and V_5. The remaining terminal voltages V_1, V_6 and V_7 are the learnable controls. Learnable scaling factors and bias parameters are added to each of the four input voltages. In the search for the desired functionality with the trained DNN, a cost function is used that promotes separation of output currents for the 16 different pixel features (see Methods).

The main problem faced in the feature mapping is that noise and instabilities of various origin lead to a distribution in the measured output current when a certain pixel feature is presented to the device multiple times (Fig. 6a). For reliable mapping, the separation of the output currents for the different features should sufficiently exceed the width of this distribution. The functionality obtained from the DNN search satisfies this criterion (Fig. 6b). In multiple verification measurements, the different features are presented subsequently to the device for 0.1 s. The result of a specific measurement is shown together with the DNN prediction (Fig. 6b, left). Histograms of the complete set of measurements are also shown (Fig. 6b, right); these involve 1,000 measurement points per feature obtained from a total of 10 measurements at 1,000 Hz spread over the course of 1 minute. A set of decision boundaries (Fig. 6b, dashed lines) was determined by using a Naive Bayes classifier that can be used to separate the features (see Methods and Supplementary Fig. 6). With these boundaries, 99.94% of the measured data are classified correctly.

Conclusions

We have proposed a generally applicable deep-learning approach to realizing desired functionality in complex multi-terminal nanoelectronic devices. The method involves the training of a DNN model that emulates the multidimensional input–output characteristics of the device, followed by gradient descent on selected control parameters of the DNN to find the desired functionality. The set-up of the DNN model, which involves input–output data collection and training, needs to be done only once per device. After the DNN model set-up, a variety of functionalities can be searched for with the model without further experimentation on the device. We demonstrated the method for nanoelectronic devices consisting of boron dopants in silicon (Si:B) that are contacted by eight terminals. Owing to the high efficiency of input evaluation by a DNN, we were able to find all of the Boolean functionalities almost two orders of magnitude faster than by performing physical measurements in combination with an evolutionary algorithm[6]. In addition, we were able to readily realize a more complex ring classification and a 2×2 pixel feature mapping functionality, demonstrating that the devices have enhanced computational capacity[6].

We expect our approach to have a broad application range. By using standard machine learning techniques and libraries, the generality of our approach ensures the optimization of a broad range of physical systems where control parameters are available. Naturally, the choice of DNN architectures determines the reliability of the functionality prediction and varies for different physical systems (see Methods and Supplementary Figs. 2–4).

As an extension of the present work, we intend to explore the coupling of many devices to create hierarchical structures with many more inputs, outputs and control parameters. Finding a desired complex functionality solely by experimentation will then become increasingly challenging. A limitation of our approach will be the time needed to sample the input voltage space, which scales exponentially with the number of input terminals if the sampling density is kept constant. In the present study, the number of input terminals is seven. If the sampling speed can be increased to 100 MHz[6] instead of the present 50 Hz, while keeping the same sampling density, the number of input terminals could be doubled to about 14 without increasing the sampling time. For many more input terminals, a modular approach can be used in which DNN models are created for separate modules with a still manageable number of terminals. The response of the total system can then be modelled by coupling the DNNs of the separate modules.

Our approach can also be valuable for the tuning of quantum dot architectures[29–31] that are used in quantum information processing. Large systems that consist of many coupled quantum dots have a large parameter space and therefore require more sophisticated tuning approaches. Existing tuning methods are based on 'device-in-the-loop' approaches[32–34] in which measurement of the physical device is required during optimization. By contrast, parameter tuning in our approach can be performed entirely off-chip. Because of its generality, our deep-learning approach can be readily applied to these systems. Finally, we note that our approach is not restricted to nanoelectronic devices, but can be applied to any complex, tuneable, static system, such as programmable metasurfaces[35], for which functionality realization is also challenging.

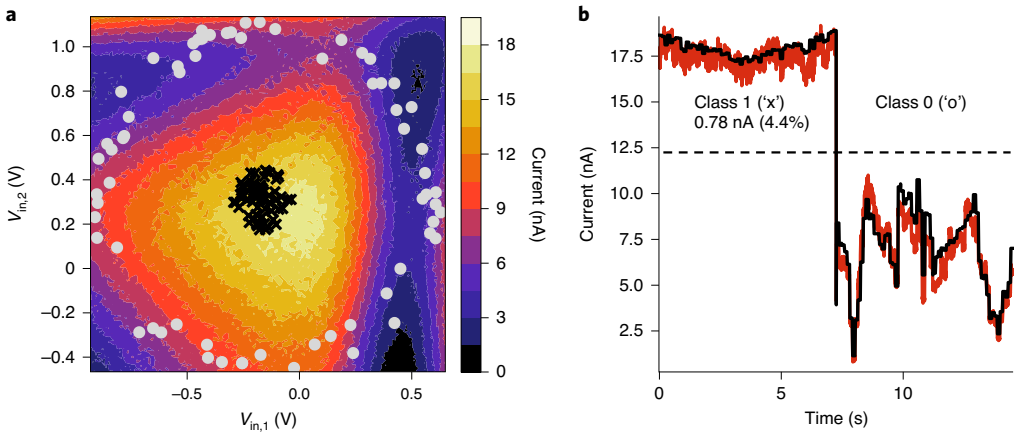

Fig. 5 | Ring classification functionality. a, The 66 outer data points (class '0', grey dots) and 66 inner data points (class '1', black crosses) to be classified. In this case V_4 and V_5 from Fig. 3b are used as data input voltages $V_{in,1}$ and $V_{in,2}$, whereas the remaining voltages are the control parameters. The heat map shows the output current of the physical device while sweeping V_4 and V_5 and keeping the predicted control voltages fixed (see Supplementary Table 6 for their values). The points undergo an affine transformation before the mapping (see Supplementary Table 7 for the scaling factor and offset voltages). **b**, Prediction by the trained DNN (black) and verification measurement (red). Grouped in classes for visualization, data points ('x' represents black crosses, 'o' represents grey dots) are each measured for 100 ms in the physical device, and between the data points the input voltages are ramped up or down to the next point with a ramping time of 10 ms to avoid transient effects. The classes are readily separable by a threshold current (dashed line).

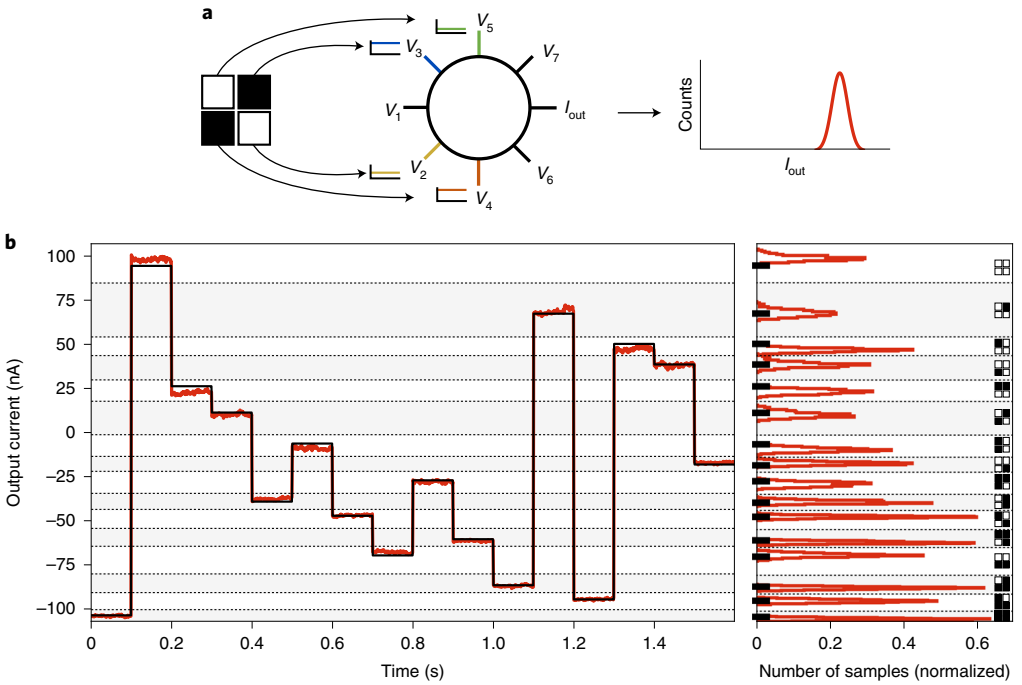

Fig. 6 | Feature mapping task. a, Schematic representation of the task. All four pixels of the feature are presented as a high (black) or low (white) input voltage to a terminal of the device, after which a histogram of the output current is obtained that should be separated from that of other features. The voltages of the remaining control terminals are optimized by the DNN model to maximize the difference between the output currents of different features (see Supplementary Tables 8, 9 for the parameters). **b**, Left, comparison of a measurement of the output current (red) to the DNN prediction (black) for the different input features. Between different presented features, the input voltages are linearly ramped over 200 ms periods to their new values, during which the current is masked. Right, histograms of the output current of 10 measurements. The dashed lines show decision boundaries that are obtained with a Naive Bayes classifier (Supplementary Table 10).

Online content

Any methods, additional references, Nature Research reporting summaries, source data, extended data, supplementary information, acknowledgements, peer review information; details of author contributions and competing interests; and statements of data and code availability are available at https://doi.org/10.1038/s41565-020-00779-y.

Received: 14 January 2020; Accepted: 9 September 2020;
Published online: 19 October 2020

References

1. Baart, T. A., Eendebak, P. T., Reichl, C., Wegscheider, W. & Vandersypen, L. M. K. Computer-automated tuning of semiconductor double quantum dots into the single-electron regime. *Appl. Phys. Lett.* **108**, 213104 (2016).
2. Kalantre, S. S. et al. Machine learning techniques for state recognition and auto-tuning in quantum dots. *Npj Quantum Inf.* **5**, 6 (2019).
3. Botzem, T. et al. Tuning methods for semiconductor spin qubits. *Phys. Rev. Appl.* **10**, 054026 (2018).
4. van Diepen, C. J. et al. Automated tuning of inter-dot tunnel coupling in double quantum dots. *Appl. Phys. Lett.* **113**, 033101 (2018).
5. Teske, J. D. et al. A machine learning approach for automated fine-tuning of semiconductor spin qubits. *Appl. Phys. Lett.* **114**, 133102 (2019).
6. Chen, T. et al. Classification with a disordered dopant-atom network in silicon. *Nature* **577**, 341–345 (2020).
7. Bose, S. K. et al. Evolution of a designless nanoparticle network into reconfigurable Boolean logic. *Nat. Nanotechnol.* **10**, 1048–1052 (2015).
8. Lykkebø, O. R., Nichele, S. & Tufte, G. An investigation of square waves for evolution in carbon nanotubes material. In *Proc. 13th European Conference on Artificial Life (ECAL)* 503–510 (MIT Press, 2015).
9. Miller, J. F., Harding, S. L. & Tufte, G. Evolution-in-materio: evolving computation in materials. *Evol. Intell.* **7**, 49–67 (2014).
10. Stepney, S. The neglected pillar of material computation. *Physica D* **237**, 1157–1164 (2008).
11. Zwolak, J. P. et al. Autotuning of double-dot devices in situ with machine learning. *Phys. Rev. Appl.* **13**, 034075 (2020).
12. Lennon, D. T. et al. Efficiently measuring a quantum device using machine learning. *Npj Quantum Inf.* **5**, 79 (2019).
13. Durrer, R. et al. Automated tuning of double quantum dots into specific charge states using neural networks. *Phys. Rev. Appl.* **13**, 054019 (2020).
14. Lapointe-Major, M. et al. Algorithm for automated tuning of a quantum dot into the single-electron regime. *Phys. Rev. B* **102**, 085301 (2020).
15. Darulová, J. et al. Autonomous tuning and charge-state detection of gate-defined quantum dots. *Phys. Rev. Appl.* **13**, 054005 (2020).
16. LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. *Nature* **521**, 436–444 (2015).
17. Hornik, K., Stinchcombe, M. & White, H. Multilayer feedforward networks are universal approximators. *Neural Netw.* **2**, 359–366 (1989).
18. Werbos, P. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* PhD dissertation, Harvard Univ. (1974).
19. Rumelhart, D. E., Hinton, G. E. & Williams, R. J. *Learning Internal Representations by Error Propagation* Report 8506 (Institute for Cognitive Science, University of California, San Diego, 1985).
20. LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **86**, 2278–2324 (1998).
21. Ghiringhelli, L. M., Vybiral, J., Levchenko, S. V., Draxl, C. & Scheffler, M. Big data of materials science: critical role of the descriptor. *Phys. Rev. Lett.* **114**, 105503 (2015).
22. Kalinin, S. V., Sumpter, B. G. & Archibald, R. K. Big–deep–smart data in imaging for guiding materials design. *Nat. Mater.* **14**, 973–980 (2015).
23. Butler, K. T., Davies, D. W., Cartwright, H., Isayev, O. & Walsh, A. Machine learning for molecular and materials science. *Nature* **559**, 547–555 (2018).
24. Carrasquilla, J. & Melko, R. G. Machine learning phases of matter. *Nat. Phys.* **13**, 431–434 (2017).
25. Arsenault, L.-F., Lopez-Bezanilla, A., von Lilienfeld, O. A. & Millis, A. J. Machine learning for many-body physics: the case of the Anderson impurity model. *Phys. Rev. B* **90**, 155136 (2014).
26. Cover, T. M. Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition. *IEEE Trans. Electron. Comput.* **EC-14**, 326–334 (1965).
27. Miller, J. F. & Downing, K. Evolution in materio: looking beyond the silicon box. In *Proc. 2002 NASA/DoD Conference on Evolvable Hardware* 167–176 (IEEE, 2002).
28. Rosenblatt, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychol. Rev.* **65**, 386–408 (1958).
29. Li, R. et al. A crossbar network for silicon quantum dot qubits. *Sci. Adv.* **4**, eaar3960 (2018).
30. Hill, C. D. et al. A surface code quantum computer in silicon. *Sci. Adv.* **1**, e1500707 (2015).
31. Veldhorst, M., Eenink, H. G. J., Yang, C. H. & Dzurak, A. S. Silicon CMOS architecture for a spin-based quantum computer. *Nat. Commun.* **8**, 1766 (2017).
32. van Esbroeck, N. M. et al. Quantum device fine-tuning using unsupervised embedding learning. *New J. Phys.* https://doi.org/10.1088/1367-2630/abb64c (2020).
33. Moon, H. et al. Machine learning enables completely automatic tuning of a quantum device faster than human experts. *Nat. Commun.* **11**, 4161 (2020).
34. Darulova, J., Troyer, M. & Cassidy, M. C. Evaluation of synthetic and experimental training data in supervised machine learning applied to charge state detection of quantum dots. Preprint at https://arxiv.org/abs/2005.08131 (2020).
35. Tsilipakos, O. et al. Toward intelligent metasurfaces: the progress from globally tunable metasurfaces to software-defined metasurfaces with an embedded network of controllers. *Adv. Opt. Mater.* **8**, 2000783 (2020).

Methods

Data sampling. To perform the measurements, we use a battery-powered electronics rack that comprises voltage sources and *I*–*V* converters for low-noise measurements. All of the measurements are automated with Python by using Ni-DAQmx software v.0.5[36] and our in-house SkyNEt framework[37]. The temperature (77 K) is fixed by dipping the device in a dewar that contains liquid nitrogen.

The training and test data are sampled by using sinusoidal (Boolean logic and ring classification functionality) or triangular (2×2 pixel feature mapping) modulated input signals to minimize capacitive transient effects. We used both input signals to test different input distributions (Supplementary Table 2). The frequencies, amplitudes and offsets of these functions for each input terminal are given (Supplementary Table 1). We choose frequencies that are proportional to the square root of prime numbers. The ratio of any two frequencies is then irrational, which guarantees a good coverage of the seven-dimensional input voltage space and prevents any recurrence of voltage combinations. To ensure that training and test data do not overlap, the training data set is generated by using modulation functions without phase shift, whereas the test set is generated with a phase shift of 1 rad (~57°). The training and test data sets comprise about 3×10^6 and 5.4×10^5 input–output pairs, respectively. The sampling occurs at a frequency of 50 Hz, where one in three points is added to the data set. For the generation of the sinusoidal and triangular voltage modulation functions, a National Instruments (NI) 9264 voltage sourcing module is used in combination with cDAQ-9171 and cDAQ-9178 chassis. Output currents are measured after an *I*–*V* conversion with either an NI USB-6216 device or an NI 9202 voltage measuring unit in combination with the cDAQ-9178 chassis.

DNN architectures. The general methodology to determine the hyperparameters of DNN models, especially regarding the architecture, is still an open question in the field of deep learning, but suggestions exist to guide the construction of suitable DNN models. If there are no previous examples of network models for the system at hand, it is advisable to start with simple models, for example linear models or shallow networks, and assess their ability to predict the behaviour of the given system. When designing the model, one should consider the amount of data available for the training and validation of the models, for which no clear guidelines exist. However, as a rule of thumb, if simple models readily overfit the data, the focus should be more on data acquisition or on other modelling methods that are suitable for small data sets. Given a suitable amount of data, it is best practice to explore architectures with different design choices, such as depth, width and activation function. At the beginning of the exploration, choosing a width for the hidden layers that is larger than the input dimension often improves performance. Increasing the depth of the DNN usually boosts performance at the beginning, but once this improvement becomes marginal, the width of the hidden layers should be increased. The aim should be to find the architecture with the best performance on the validation data set that also generalizes well with test data. (An example of this procedure is given in Supplementary Fig. 1.) If overfitting is observed, the DNN can be regularized to improve performance. If the cost levels off at an acceptable value for different architectures, it is advisable to take the simplest model to avoid computational overhead. If further efficiency considerations are important after finding the best DNN model, the computational complexity of the model may be optimized through parameter quantization and pruning, low-rank factorization, or knowledge distillation approaches[38]. There are, however, important considerations when making a trade-off between accuracy and computational or model complexity. In general, sacrificing the prediction accuracy will result in greater efforts in searching for functionality, because the probability of false positives and negatives will increase (Supplementary Figs. 2–4). We refer to Supplementary Section 2 for a detailed discussion on the consequences of this trade-off.

DNN training. The neural network consists of seven inputs and one output, which correspond to the input and output terminals of the physical device, respectively. We use a fully connected feedforward network that consists of five hidden layers with 90 nodes (see Supplementary Section 2 for the choice of this architecture) and a ReLU (rectified linear unit) activation function. Training is done by stochastic gradient descent on the mean squared error for 3,000 epochs with a learning rate of $\eta=10^{-5}$ and a mini-batch size of 128. The training data set is split into a set (90%) used for the training and a set (10%) used for the validation. The validation data set should not be confused with the test data set. The validation set is used for hyper-parameter optimization and to prevent overfitting, whereas the test set of unseen data is used to estimate the generalization error. For optimization, Adam[39] is used with $\beta_1=0.9$ and $\beta_2=0.75$. All of the hyperparameters are explored to obtain the lowest possible validation error. Our DNN model is implemented by using PyTorch v.1.1.0[40].

Cost functions. The cost function to obtain Boolean logic and ring classification functionality is given by

$$E(y,z)=(1-\rho(y,z))/f\left(\left(y_{\mathrm{sep}}-q\right)/p\right),$$

where $y=(y_1,y_2,\ldots,y_n)$ are the actual currents, $z=(z_1,z_2,\ldots,z_n)$ are the targeted binary current levels ($n=4$ for Boolean logic and $n=132$ for ring classification), $\rho(y,z)$ is the Pearson correlation coefficient and $f(x)=1/(1+e^{-x})$ is the standard logistic function. The current values q and p control the desired separation and are chosen to be 3 nA and 5 nA, respectively. The value y_{sep} represents the minimum separation between the high and low labelled data. For the data labelled as class '1' (high output) the lowest predicted current is taken, and for the data labelled as class '0' (low output) the highest predicted current is taken; the difference is used to obtain y_{sep}. The correlation function promotes similarity between the targeted and actual outputs, whereas the logistic function promotes separation of the two classes.

Although the binary cross-entropy loss is most often used for binary classification tasks, its use would require the mapping of the output of our device to the posterior probability over the targets. This requires a linear readout to be learned together with the control voltages. Here, we opt for a cost function that avoids the introduction of extra parameters. We designed our cost function to be a differentiable function of the control parameters that reflects the characteristics of the fitness function that was used previously to find Boolean functionality[6].

For the 2×2 pixel feature mapping, we use the cost function

$$E(y)=-\sum\nolimits_i f\left(\left|y_i-y_{\mathrm{NN}(i)}\right|/p\right),$$

where y_i is the current for feature i ($i=1,\ldots,16$) and $y_{\mathrm{NN}(i)}$ is the 'nearest-neighbour' current (NN(i) is the feature with current closest to y_i). The logistic function promotes an initial increase in current separation and leads to a saturation for a sufficiently large separation. We take p to be 2 nA, which leads to saturation of the cost function for separations above 10 nA.

Control parameters. The Boolean logic gates consist of four distinct combinations of two binary states for the input terminals. The voltages that represent these states are taken to be −1.2 V and 0.6 V, which correspond to '0' and '1', respectively. The voltages of the remaining five terminals are the learnable control parameters. The input data set of the Boolean logic gates is expanded such that each input combination is represented 100 times, thereby leading to a total of 400 data points in the training set. During optimization, these data points are presented randomly as inputs to the DNN. Stochastic gradient descent is used with a mini-batch of 100 data points and a learning rate of $\eta=0.08$. A single optimization session for a logic gate consists of at most 600 epochs, with an early termination if in the previous 150 iterations there has not been a substantial reduction in error. To prevent the learnable parameters from deviating too much from the voltage range of the training data set, the parameters are regularized with an L1 norm outside this range. To obtain the best results, we re-initialized training 10 times per logic gate. The predicted control voltage values with the lowest error (Supplementary Table 5) are the ones taken in the verification.

For the ring classification problem, 132 input data points are used (Fig. 5a). The hyperparameters used for optimization are the same as for the Boolean logic gate optimization, except for the number of initializations and the maximum number of iterations per initialization, which are 20 and 800, respectively. The 66 outer points are generated uniformly and randomly in a ring with radii of 0.5 V and 0.6 V, and the 66 inner points in a circle with a radius of 0.1 V. The coordinates (x_1, x_2) of these points are transformed to input voltages by

$$V_{\mathrm{in},i}=x_i V_{\mathrm{scaling}}+V_{\mathrm{offset},i},$$

where V_{scaling} is a scaling voltage, taken to be equal for the two input electrodes, and $V_{\mathrm{offset},i}$ are voltage offsets. These parameters define an affine transformation of the data and are, next to the control voltages, the learnable control parameters. The control voltages for the ring classification functionality are given in Supplementary Table 6, and the scaling voltage and offsets of the two input voltages are given in Supplementary Table 7. Note that, in this example, the input data terminals are 4 and 5, which are different from the Boolean logic gates (terminals 2 and 3).

For the feature mapping functionality (Fig. 6), we map the 16 combinations of four-dimensional binary patterns to 16 distinguishable current output levels. The binary values $x_{\mathrm{base}}=0$ for 'high' and −1 for 'low' are mapped to input voltages

$$V_{\mathrm{in},i}=x_{\mathrm{base}}V_{\mathrm{scaling},i}+V_{\mathrm{offset},i},$$

where again $V_{\mathrm{scaling},i}$, now taken to be different for the four input electrodes, and $V_{\mathrm{offset},i}$ are added to the learnable control parameters. The final functionality is the best result obtained after 500 random initializations of the control parameters and training for 5,000 epochs with a mini-batch of four. The resulting control voltages are given in Supplementary Table 8, and the scaling factors and offsets of the four input voltages are given in Supplementary Table 9.

Pixel feature decision boundaries. The set of decision boundaries for the pixel features (Fig. 6b) is determined by using a Naive Bayes classifier with Gaussian data approximation, which optimizes the posterior probability. The collective measured data (Fig. 6b, right) are used as training set for finding the decision boundaries. Using standard Bayesian notation, a new data point x is assigned to class k' for which

$$P(k')P(x|k')=\max_k[P(k)P(x|k)].$$

Here, $P(k)$ is the prior probability, which is equal to the fraction of data points that belong to class k in the training set, and $P(x|k)$ is the probability density function of class k in the training set. In our case the classes have an equal prior probability, so $P(k) = P_0 \forall k$. The probability density function $P(x|k)$ of each class $k = 1,\ldots,16$ in the training set is assumed to be Gaussian in our case (see histograms in Fig. 6b, right). Therefore, in our case a new data point x is assigned to the class k that has the highest probability density for that value of x. Graphically, this means that the decision boundaries are located at the crossings of Gaussian fits to the histograms (Fig. 6b, right), as shown in Supplementary Fig. 6. The currents corresponding to the decision boundaries are given in Supplementary Table 10. The classification is done with the Gaussian Naive Bayes module of the scikit-learn v.0.22.1 package[41].

Data availability

Data are available from the public repository https://data.4tu.nl at https://doi.org/10.4121/12884804.

Code availability

The custom computer code used here is available under the GNU General Public License v3.0 at https://github.com/BraiNEdarwin/SkyNEt.

References

36. *NI-DAQmx Python documentation* (National Instruments Corp., 2017) https://nidaqmx-python.readthedocs.io/en/latest
37. *SkyNEt library* (Darwin team of the NanoElectronics group, Univ. of Twente, 2020) https://github.com/BraiNEdarwin/SkyNEt
38. Cheng, Y., Wang, D., Zhou, P. & Zhang, T. A survey of model compression and acceleration for deep neural networks. *IEEE Signal Process. Mag.* https://arxiv.org/abs/1710.09282 (2017).
39. Kingma, D. P. & Ba, J. L. Adam: a method for stochastic optimization. In *Proc. 3rd International Conference for Learning Representations (ICLR).* https://arxiv.org/abs/1412.6980 (2015).
40. Paszke, A. et al. PyTorch: an imperative style, high-performance deep learning library. In *Proc. 33rd Conference on Neural Information Processing Systems (NeurIPS)* 8024–8035 (2019).
41. Pedregosa, F. et al. Scikit-learn: machine learning in Python. *J. Mach. Learn. Res.* **12**, 2825–2830 (2011).

Acknowledgements

We thank B. J. Geurts, U. Alegre Ibarra, B. de Wilde and L. J. Knoll for fruitful discussions. We are grateful to U. Alegre Ibarra for reading the manuscript carefully and providing useful input. We thank M. H. Siekman and J. G. M. Sanderink for technical support. We acknowledge financial support from the University of Twente, the Dutch Research Council (NWA Startimpuls grant no. 400-17-607) and the Natuurkunde Projectruimte (grant no. 680-91-114).

Author contributions

H.-C.R.E., M.N.B. and J.T.W. performed the measurements and the DNN modelling. B.v.d.V. and T.C. fabricated the samples. H.-C.R.E., M.N.B., J.T.W., P.A.B. and W.G.v.d.W. wrote the manuscript and all of the authors contributed to revisions. W.G.v.d.W. and H.-C.R.E. conceived the project and designed the experiments with input from M.N.B. and J.T.W. W.G.v.d.W., P.A.B., H.B. and H.-C.R.E. supervised the project.

Competing interests

The authors declare no competing interests.

Additional information

Supplementary information is available for this paper at https://doi.org/10.1038/s41565-020-00779-y.

Correspondence and requests for materials should be addressed to W.G.v.W.

Peer review information *Nature Nanotechnology* thanks Matthew Dale, Gunnar Tufte and the other, anonymous, reviewer(s) for their contribution to the peer review of this work.

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