Neural networks cannot typically be trained locally in edge-computing systems due to severe energy constraints. It has, therefore, become commonplace to train them “ex situ” and transfer the resulting model to a dedicated inference hardware. Resistive memory arrays are of particular interest for realizing such inference hardware, because they offer an extremely low-power implementation of the dot-product operation. However, the transfer of high-precision software parameters to the imprecise and random conductance states of resistive memories poses significant challenges. Here, it is proposed that Bayesian neural networks can be more suitable for model transfer, because, such as device conductance states, their parameters are described by random variables. The ex situ training of a Bayesian neural network is performed, and then, the resulting software model is transferred in a single programming step to an array of 16,384 resistive memory devices. On an illustrative classification task, it is observed that the transferred decision boundaries and the prediction uncertainties of the software model are well preserved. This work demonstrates that resistive memory-based Bayesian neural networks are a promising direction in the development of resistive memory compatible edge inference hardware.

In recent years, neural network models have demonstrated human-level competency in multiple tasks, such as pattern recognition, game playing, and strategy development. This progress has led to the promise that a new generation of intelligent computing systems could be applied to such high-complexity tasks at the edge. However, the current generation of edge computing hardware cannot support the energetic demands nor the data volume required to train and adapt such neural network models locally at the edge. One solution is ex situ training: a software model is trained on a cloud computing platform and then subsequently transferred onto a hardware system that acts only to perform inference. The engine room of such inference hardware is the dot-product (multiply-and-accumulate) operation that is ubiquitous in machine learning. Non-von Neumann dot-product implementations based on nonvolatile resistive random access memory (RRAM) technologies, otherwise known as memristors, are a particularly promising path toward reducing the energy required during inference. Here, the dot product between an input voltage vector and an array of RRAM conductance states, storing the parameters of the model, can be evaluated in-memory through the physics of Ohm’s and Kirchhoff’s laws—obviating the need to transfer information on-chip.

In these systems, however, intrinsic cycle-to-cycle and device-to-device conductance variability constitute a considerable challenge. This random variability constrains the number of separable multilevel conductance states that can be achieved, preventing the high-precision transfer of the model parameters in a single programming step. To mitigate against this variability, iterative closed-loop programming schemes (also referred to as program–verify schemes) are often used, whereby devices are repeatedly programmed until their conductance falls within a discretized window of tolerated error. However, such approaches entail costly circuit overheads as well as energy and time during model transfer. Other approaches propose to use multiple one-transistor-one-resistor (IT1R) structures in parallel at each array cross-point, which, although allow for a higher precision in the transferred weight, entail additional costs in area and transfer energy. Other approaches propose to sidestep intrinsic randomness altogether through the quantization of conductance levels into one of either a low-conductance state (LCS) or a high-conductance state (HCS) in binarized neural networks. However, such models require a significantly higher number of neuron elements and model parameters to approach the performance of conventional neural networks.

The reality is that the programming of resistive memory is an inherently random process, and the devices, therefore, are not well suited to being treated as deterministic quantities.
Fortunately, however, this randomness follows stereotyped probability distributions and allows resistive memory devices to be instead yielded as physical random variables. Previous work has used the probability distributions that emerge from the volatile random switching properties of magnetic RRAM and stochastic electronic circuits for example, to perform Bayesian inference.

In this article, by working within this framework of probabilistic programming and Bayesian modeling, we propose and experimentally demonstrate an approach for the ex situ training and subsequent transfer of a Bayesian neural network into the nonvolatile conductance states of a resistive memory-based inference hardware. In Bayesian neural networks, like resistive memory conductance states, model parameters are not single high-precision values but probability distributions—suggesting a more natural pairing of algorithm and technology. In this setting, the objective is no longer to precisely transfer a single parameter from a software model to the corresponding device in a resistive memory array but to transfer a probability distribution from the software model into a distribution of device conductance states.

In this work, we first propose to use an expectation-maximization algorithm to decompose a probability distribution into a small number of random variable components, each corresponding to the cycle-to-cycle conductance distribution of RRAM programmed under a given programming current. We then experimentally demonstrate, with an array consisting of 16 384 fabricated hafnium dioxide 1T1R structures, that these random variable components can be used to transfer the original probability distribution into the conductance states of a column of RRAM devices. We show that this approach can be leveraged to achieve the transfer of a full Bayesian neural network in a single programming step. We also describe an RRAM-based hardware capable of storing, and performing inference with, such a Bayesian neural network. Finally, a Bayesian neural network model is trained ex situ and transferred using the proposed technique to an experimental array. The transferred model is used for inference in a simple illustrative classification task where the decision boundaries that were learned ex situ are seen to be well preserved in the model transferred into the inference hardware.

Resistive memory devices store information in their nonvolatile conductance states. These conductance states can be programmed by applying voltage or current waveforms over the device in a fashion specific to the type of memory technology. Here, we consider an oxide-based resistive random access memory (OxRAM) composed of a thin film of hafnium dioxide sandwiched between a top and a bottom electrode of titanium and titanium nitride. By applying a positive SET voltage between the top and bottom electrodes, a filament of conductive oxygen vacancies is instantiated within the oxide between the electrodes. This filament can thereafter, by applying a negative RESET voltage, be disrupted. By applying successive SET and RESET pulses, RRAM devices are cycled between their HCS and LCS. We have cointegrated such devices in a standard 130 nm complementary metal–oxide-semiconductor (CMOS) process to realize a fabricated array of 16 384 RRAM devices, which we use as our experimental platform throughout this article. Each device is connected in series with an n-type transistor, realizing a 1T1R structure. This structure allows each device to be individually selected for reading and programming.

In RRAM, the random mechanisms governing the distribution of vacancies within the oxide dictate that, between successive programming operations, the device will assume a different conductance state from the previous one. If a device is repeatedly cycled under the same programming conditions, a normally distributed cycle-to-cycle conductance variability emerges for the HCS. In addition, the median conductance of this normal distribution can be determined by limiting the current that flows during the SET operation. As an example, the cycle-to-cycle conductance variability distributions of a single device using three different SET programming currents are shown in Figure 1a. Notably, the standard deviation of each normal distribution is intrinsically tied to the median of the distribution: the standard deviation reduces, as the conductance median increases. This result is summarized by plotting the average relationship between the cycle-to-cycle conductance standard deviation and the median for the full population of 16 384 devices in

Figure 1. a) Three cycle-to-cycle conductance variability probability distributions for a single device programmed 100 times under three different SET programming currents. The raw data are plotted as three histograms, which have each been fitted with a normal distribution (dashed line). b) The median standard deviation of the cycle-to-cycle and device-to-device conductance variability distribution of all 16 384 devices is plotted as a function of the medians of all device conductance variability distributions for a sweep of the SET programming current. The relationship can be approximated with a linear function (dashed line).
the 1T1R array, which can be approximated with a linear function (Figure 1b). Each RRAM device is, therefore, a normal physical random variable; the conductance states that result from SET operations are analogous to drawing samples from a normal distribution with a median and standard deviation determined by the programming current.

Bayesian neural networks are variants of conventional neural networks, whereby parameters are not single values, but probability distributions. The distribution of each parameter encapsulates the uncertainty in its estimation, which allows for a model to avoid overfitting, given, for example, a small training dataset or noisy sensory observations.

Therefore, the challenge is to transfer these software-based probability distributions into a plurality of device conductance states on an RRAM-based inference hardware. The fundamental insight of this article is that, because RRAM conductance states are also probability distributions, they owe themselves more naturally to the transfer of ex situ trained Bayesian neural network models than deterministic ones.

We propose that the probability distribution of each Bayesian neural network parameter can be approximated by a linear combination of weighted normal random variable components—determined using a Gaussian mixture modeling approach. In a Gaussian mixture model, each of the K Gaussian distributions, also referred to as normal random variable components, is characterized by a median, a standard deviation, and a weighting factor. These three parameters per component are updated iteratively using an algorithm called expectation–maximization until a mixture of components is found that best "explains" the target parameter distribution.

As shown in the previous section, OxRAM devices are normal physical random variables in the HCS (Figure 1a). However, although the median of each RRAM random variable component can be freely determined by the SET programming current (Figure 1a), the standard deviation is intrinsically tied to this value (Figure 1b). This requires that, instead of treating the standard deviation of each component as free parameters during the expectation–maximization algorithm, its value must be assigned based on the known relationship with the median (here, the equation in Figure 1b). This may require, for example, additional circuitry on a practical chip to perform an initial calibration step owing to the die-to-die variability that exists across chips on a wafer as well as between wafers.

We apply this technique to decompose the single target probability distribution plotted in green in Figure 2a into K physical random variable components. These components, determined through expectation–maximization, can then be used to program experimentally a column of N RRAM memory cells, such that the distribution of conductance states in the column approximates that of the target distribution. This result is achieved by programming subsets of devices in the column with a SET programming current, such that their conductance states are sampled from the Gaussian corresponding to each component. The number of devices programmed per component is equal to the nearest integer value resulting from the multiplication of the total number of available devices by its weighting factor. For this target distribution, it was found that five normal components (K = 5) were required to well approximate the target distribution. This result was obtained by performing the expectation–maximization algorithm over a sweep of K and observing at which value of K the resulting log-likelihood of the mixture saturated—as plotted in Figure 2b. The five resulting RRAM random variable components are superimposed over the original target distribution in Figure 2a. These five components are then used experimentally to program a column of 1024 1T1R RRAM devices as described—the number of devices programmed per component is specified in the caption of Figure 2a. The resulting probability distribution, plotted as a histogram in Figure 2a, is seen to well approximate that of the original target distribution. This also suggests that the linear approximation of the relationship between conductance median and standard deviation, which has a nonnegligible error at the conductance extremities, is not detrimental.

To quantify the closeness of the approximation transferred to the hardware, we evaluate the Kullback–Leibler (KL) divergence from the transferred to the target distributions over a range of column sizes. The resulting mean KL divergence, over ten experimental transfers, is plotted in Figure 2c for an increasing number of RRAM cells per column. The KL divergence reduces rapidly, as the number of devices in the RRAM column is increased, consistent with the law of large numbers.

Before applying the presented technique to the transfer of a full Bayesian neural network model, we first describe how to perform the ex situ training of an RRAM-based Bayesian neural network, and how the parameters from this software model can be represented using an array of resistive memory devices.

In the Bayesian framework, training is typically performed with Markov chain Monte Carlo (MCMC) sampling or using variational inference algorithms. In this article, we use the No-U-Turn sampler (NUTS) MCMC algorithm. In contrast to gradient-based approaches, which result a deterministic locally optimal model, NUTS MCMC results in a collection of sampled models, each with their own parameters (synaptic weights and biases). The distribution of each learned parameter, in other words, the distribution of parameters over all of the sampled models, can then be transferred to a distribution of device conductances in a column of RRAM cells. In contrast to deterministic models, which generally require a single device per parameter, the use of a distribution comprising multiple devices per parameter allows uncertainty to be incorporated into its estimation. The ability to represent uncertainty in this manner is an advantage of Bayesian models, permitting them to account for factors, such as sensory noise, and small training dataset size as well as propagating uncertainty into their output predictions.

One neuron in an RRAM-based Bayesian neural network can be realized, as shown in Figure 3b. The neuron receives input synapses from M (here, three) neurons in the previous network layer (Figure 3a)—each connecting to one of its three columns. The distribution of the three input synaptic parameters is each stored in a column of size N and, therefore, necessitates an N × M array of 1T1R structures per neuron.

By applying a voltage vector V across these M columns, corresponding to the activations of the neurons in the previous network layer or the input data for neurons in the first layer, a current equal to \( V \cdot g_c \) flows out of each array row and into a neuron circuit. As the conductance values of \( g_c \) are on the order of microsiemens, the neuron circuit must first multiply this current value by a scaling factor \( S \) and then apply an activation function.
to the scaled quantity, resulting in a neuron output voltage
\[ z_n = h(S(V \cdot g_n)) \]. This voltage can then, in turn, be applied to a column of each of the neuron arrays in the next layer. The distribution of these \( N \) neuron activation voltages, \( z \), constitutes the output distribution of the neuron. In this article, we use the hyperbolic tangent activation function for all neurons besides those in the output layer where the softmax function is used. Use of the softmax function at the output allows the likelihood of the model to be evaluated as a categorical random variable during training, such that it can be applied to multiclass datasets.

In practice, as each parameter distribution can assume positive and negative values, each model parameter should be described by the difference between positive and negative distributions: \( p(g) = p(g_+) - p(g_-) \) (Figure 3b). Therefore, during MCMC sampling, the parameters, which are sampled, are \( p(g_+) \) and \( p(g_-) \) and not \( p(g) \) directly. In addition, each neuron of the Bayesian neural network requires a bias distribution \( p(g_b) \), that can be realized with an extra column of devices, identical to the others, to which a constant voltage \( V_b \) is applied.

One further technological constraint must be considered. Each RRAM device has a limited conductance range; in the technology applied here, extending approximately from 20 to 120 \( \mu \)S (Figure 1b). As a result, the sampled distributions of each parameter, \( p(g_+) \) and \( p(g_-) \), of the Bayesian neural network must be bounded within these limits during the ex situ training. Fortunately, in the Bayesian framework, such a bounding can
be achieved naturally by placing an appropriate prior distribution over each parameter. To account for this, we, therefore, place a normal prior over each parameter, with a median of 80 μS and a standard deviation of 20 μS, such that the sampled distributions exist within the limited conductance range.

We now combine the ideas of the two previous sections and present an approach to achieve the transfer of an ex situ trained Bayesian neural network onto the RRAM-based hardware shown in Figure 4a.

The detailed methodology of the transfer is presented in Note 1 and Figure S1, Supporting Information, such that, here, we present only the core principles.

To transfer the ex situ trained Bayesian neural network to the inference hardware, the software model resulting from NUTS MCMC is required to be processed in two core steps. First, the expectation–maximization algorithm is applied to each parameter of the software model to decompose each parameter into K components. The identified components for each parameter are then used to quantize the software model by setting each of sampled values equal to the closest normal component median. Second, the quantized software model is then transferred in a row-wise fashion to the RRAM-based hardware. Each RRAM device is programmed with a SET current, such that the device assumes a conductance value sampled from a normal distribution centered on the corresponding software value. It is important to perform the transfer row-wise, because each sample produces a different activation voltage that must be propagated to the corresponding sample in the next layer. At the output layer, then, a separate memory structure is required to temporarily store each of the output neuron activations, which result from the independent forward propagation of neuron voltage activation distributions through the subsequent network layers, finally resulting in an activation distribution per output neuron. These output distributions can then be used to make a prediction regarding as to what output neuron class the input data point belongs. In addition, the standard deviation of each prediction distribution can be calculated and used to quantify the uncertainty in the prediction of each output neuron.

As the pre-synaptic distributions of each of the neurons in a Bayesian neural network are stored in multiple rows (i.e., multiple samples), but the physical connections between each neuron consist only of single metal wires, inference with a Bayesian neural network must be performed one row (sample) at a time—because each sample produces a different activation voltage that must be propagated to the corresponding sample in the next layer. At the output layer, then, a separate memory structure is required to temporarily store each of the output neuron activations, which result from the independent forward propagation of an input data point through each of these transferred samples. In this fashion, after all of the rows have been read in an inference, the prediction distribution of each output neuron is readily available. To achieve this, we propose, in Figure 4a, that each neuron array contains only a single neuron circuit that is multiplexed between each of the N rows sequentially. By applying voltage pulses to the gates of the devices in only one row, while

Figure 3. a) A feedforward neural network of three neurons (M = 3) in the first layer and three neurons in the second layer. For the case of a Bayesian neural network, the synapses (for example, highlighted in green) and neurons (for example, highlighted in blue) are described by probability distributions. The RRAM-based realization of the Bayesian neuron and synapses enclosed in the gray-dashed triangle is shown in b). b) A proposed structure for realizing a Bayesian neuron (and the synapses fanning into it) based on an N × M array of resistive memory. The distribution of conductance states of the devices in a column corresponds to the distribution of a synaptic parameter (for example, highlighted in green). Each row of the array uses devices that code for positive (g+) and negative (g−) values that enables each parameter to be positive or negative. The inputs to the columns are the output voltages generated by the M neurons in the previous layer. As a result of these input voltages, two currents will flow out of each row and into a neuron circuit, which subtracts them and then evaluates an activation function. This activation produces an output voltage as a function of this current that can then be applied to the column of neuron arrays in a subsequent layer. The distribution of the N output voltages (blue probability distribution) is the output distribution of the neuron.
grounding the others, this multiplexing can be achieved cheaply and without a dedicated multiplexing circuit. In addition, the use of a shared neuron circuit also reduces the required circuit overhead to implement the activation function or to perform any required analogue-to-digital conversions by a factor of $N$. For example, by applying the red pulses in Figure 4a to row $N = 0$ of all of the neuron arrays simultaneously at time $t_0$, each neuron in the output layer will produce a voltage activation, $z_0$. In other

Figure 4. a) Proposed hardware realization of feedforward layers in the RRAM-based Bayesian neural network shown in b). The output of three hidden-layer neuron arrays, corresponding to neurons one, two, and eight in b), are connected to the inputs of three columns of RRAM of another neuron array, neuron one in the output layer in b). As a function of the input data feature voltages (from the red-colored neurons in the first layer), the hidden layer neurons will produce activation voltages that are, in turn, applied over the columns of the output layer neurons causing the output layer neurons to produce activation voltages. This forward propagation of voltage continues for an arbitrary number of network layers until reaching the output layer. By sequentially applying gate voltage pulses to each row of all the arrays—the red pulses at $t_0$, the green pulses at $t_1$, and, finally, the blue pulses at $t N / C_0$—output neuron one will sequentially produce voltage activations $z_0, z_1, \text{and } z_{N-1}$. The distribution of all activations, $p(z)$, gives rise to an output distribution of neuron one. b) (Center) A single hidden-layer feedforward Bayesian neural network. Circles and lines in bold correspond to the neuron arrays and connections shown in part (a). (Left) The probability density histograms and kernel density estimates for a synaptic parameter (green) using 16, 128, and 1024 memory cells per column. (Right) The predictive probability contours of neuron one (recognizing points from the red moon) and neuron two (blue moon) for 16 (right), 128 (centre), and 1024 (left) memory cells per column. Each of the red and blue moons data points is described by two feature voltages that are applied as inputs to the columns of the green neuron arrays.
words, these output activations result from the forward propagation of the input through the devices in row $N = 0$ of each array only. Thereafter, applying the green pulses at $t_1$, the outputs $z_1$ will result from the forward propagation through the devices at rows $N = 1$ and so on. By proceeding in this fashion up until row $N - 1$, an output distribution $p(z)$ will be available for each output neuron at $t_{N-1}$. The mean value and standard deviation of this distribution can be used to, respectively, make a prediction and quantify prediction uncertainty.

To demonstrate this technique, we perform the ex situ training of the Bayesian neural network shown in Figure 4b. We apply it to an illustrative example in the generative moons classification task. Each of the two output neurons of the network must learn a nonlinear decision boundary that separates its respective class of noisy data points from the other. To evaluate the transfer of the Bayesian neural network model, we perform a hybrid hardware/software experiment. After termination of NUTS MCMC, the normal random variable components required for all of the model parameters are identified using expectation-maximization. Then, 1024 devices in the experimental RRAM array are programmed, using the corresponding SET programming currents based on the median value of each of these random variable components. The resulting conductance values are then used to build up a computer model of the proposed hardware shown in Figure 4a to perform an inference. This is required because the experimental 1T1R array features parallel running source and bit lines, instead of orthogonal source and bit lines, such that devices are addressed for read or programming individually. Each RRAM cell of this computer model is randomly assigned one of the 1024 transferred conductances, which resulted from the SET programming current that would have been used to program the equivalent device on the physical array. Examples of the resulting distribution transferred to the synaptic parameter highlighted in green in Figure 4b are plotted for 1024, 128, and 16 rows. On average, based on the measured SET programming currents, the programming energy required to perform the transfer of the full Bayesian neural network model to the array was $1.37 \mu\text{j}, 172 \text{nJ}$, and $21.5 \text{nJ}$ for the models based on 1024, 128, and 16 rows, respectively.

Upon performing inference with a hybrid hardware/software model, the decision boundaries for each of the two output neurons for the model transferred to the 1024, 128, and 16 row arrays, shown in Figure 4b, arise. The output neurons appear, in all situations, able of discerning the underlying structural separation between the two types of data point that was learned in the software model. The probability contours of the two output neurons are largely similar for the case of 1024 and 128 rows, whereas those for 16 rows appear more erratic. Despite this appearance, however, the boundaries drawn at the interface of the two moons with $N = 16$ rows still capture the fundamental curvature of their division. Based on the read currents of the programmed devices, the energy required to read all of the device conductances during inference was $110 \text{nJ}, 13.7 \text{nJ}$, and $1.72 \mu\text{J}$ for the models transferred to the 1024, 128, and 16 row arrays, respectively. However, that is important to note that the energy required by read circuitry, analogue-to-digital and digital-to-analogue conversions, and circuits for implementing the neuron activation functions has not been considered and would lead to a considerable increase in these values depending on design choices.

The prediction uncertainty of each of these transferred Bayesian models is plotted in Figure S2. Supporting Information. This uncertainty, captured in the distribution of each synaptic parameter, naturally propagates through a Bayesian neural network to the output layer, where, as might be expected, it is seen to be greatest at the interface between the red and blue points. While the prediction uncertainty contours are largely similar for $N = 1024$ and $N = 128$, they are once again degraded for $N = 16$. In safety-critical edge inference applications, the ability of a Bayesian neural network to quantify uncertainty, with respect to deterministic models, is potentially invaluable and, perhaps, indispensable from an ethical perspective. For example, in a medical system, such as an implantable cardioverter-defibrillator, these prediction uncertainties can be leveraged to avoid the erroneous application of an electric shock to the heart, which can, in some instances, prove fatal. If the system was presented with a data point close to a noisy decision boundary (as in Figure S2, Supporting Information) or with a data point from a location in the feature space that the model had not observed during training, perhaps due to a damaged or drifting sensor, the prediction uncertainty of the model will be large. By placing a threshold on a tolerated level of prediction uncertainty, above which the system should not take action, erroneous interventions can be avoided.

In this article, we have presented, and demonstrated in a hybrid hardware/software experiment, a method for transferring an ex situ trained Bayesian neural network model onto a resistive memory-based inference hardware. Unlike previous transfer approaches, whereby iterative closed-loop programming (program–verify) schemes are used, an expectation–maximization-based approach facilitated the transfer of a Bayesian neural network in a single programming step. This is particularly important, because Bayesian neural networks use multiple devices to describe the probability distribution of each parameter. We have also found that, in the simple illustrative task addressed, despite the fact that each of the devices was programmed only once without verification, the decision boundaries of the software model were well preserved. Furthermore, it was demonstrated and discussed how the prediction uncertainty that is available in this Bayesian modeling approach could be an important facet in the ethical application of ex situ trained models in edge inference.

Going forward from this initial proposal and experimental demonstration, future work will focus on understanding how the proposed technique can scale to larger network models and to higher-complexity datasets as well as exploring further Bayesian ex situ training algorithms such as variational inference. It will also be instructive to perform a quantitative comparison between ex situ trained RRAM-based Bayesian and deterministic neural network models to understand the advantages and trade-offs between the two approaches in terms of inference accuracy, the energy and latency incurred in model transfer and inference, and the memory requirements.

Ultimately, the article proposes a new approach in the deployment of ex situ trained software models at the edge based on Bayesian neural networks. Such models offer certain advantages such as an increased compatibility with resistive memory properties as well as their ability to represent uncertainty that has important implications in ethical edge inference.
Experimental Section

The Bayesian neural network model, its ex situ training, and the NUTS algorithm were implemented using the python library PyMC3. The experiments were conducted on a 16,384 (16k) device 1T1R array cointegrated in a 130 nm CMOS process.

The measurements presented in Figure 1a,b consist of cycling each device in the 16k device array 100 times and performing a read operation after each SET for a sweep in SET programming current.

The results plotted in Figure 2a,c are obtained by programming 1024 (1k) regions of the 16k device array with SET programming currents that sampled from RRAM normal random variables with medians equal to the component medians determined through the expectation–maximization algorithm. The expectation–maximization algorithm was implemented in a custom python program. The results plotted in Figure 4b are obtained by programming 1024 (1k) regions of the 16k device array with SET programming currents determined using the expectation–maximization algorithm down each column of sampled models returned by the NUTS algorithm. The computer model of the hardware proposed in Figure 4a is constructed in a custom python program and then filled with the conductance values transferred to the 1T1R array.

Supporting Information

Supporting Information is available from the Wiley Online Library or from the author.

Acknowledgements

The authors would like to acknowledge the support of the French ANR via Carnot funding and the European Research Council (grant NANOINER, number 715872) for the provision of funding.

Conflict of Interest

The authors declare no conflict of interest.

Keywords

Bayesian machine learning, memristors, neural networks, resistive memories

Received: May 15, 2020
Revised: September 6, 2020
Published online:

1. Y. LeCun, Y. Bengio, G. Hinton, Nature 2015, 521, 436.
2. Advances in Neural Information Processing Systems 25, (Eds.: A. Krizhevsky, I. Sutskever, G. E. Hinton, In F. Pereira, C. J. C. Burges, L. Bottou, K. Q. Weinberger), Curran Associates, Inc., Red Hook 2012, pp. 1097–1105.
3. V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis, Nature 2015, 518, 529.
4. O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danilhela, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, et al., Nature 2019, 575, 350.
5. W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu, IEEE Internet Things J. 2016, 3, 637.
6. E. Strubell, A. Ganesh, A. McCallum, in 57th Annual Meeting of the Association for Computational Linguistics (ACL), Florence, Italy, July 2019.
7. D. Li, X. Chen, M. Becchi, Z. Zong, in 2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom), IEEE, Piscataway, NJ 2016, pp. 477–484.
8. X. Xu, Y. Ding, S. Hu, M. Niemier, J. Cong, Y. Hu, Y. Shi, Nat. Electron. 2018, 1, 216.
9. Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, J. Zhang, Proc. IEEE 2019, 107, 1738.
10. H. P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, K. E. Goodson, Proc. IEEE 2010, 98, 2201.
11. C. Chappert, A. Fert, F. Dau, Nat. Mater. 2007, 6, 813.
12. Q. Liu, J. Sun, H. Lv, S. Long, K. Yin, N. Wan, Y. Li, L. Sun, M. Liu, Adv. Mater. 2012, 24, 1774.
13. A. Beck, J. G. Bednorz, C. Gerber, C. Rossell, D. Widmer, Appl. Phys. Lett. 2000, 77, 139.
14. M. Hu, J. P. Strachan, Z. Li, R. Stanley Williams, in 2016 17th Int. Symp. on Quality Electronic Design (ISQED), Santa Clara, CA, USA 2016, pp. 374–379.
15. Q. Xia, J. J. Yang, Nat. Mater. 2019, 18, 309.
16. R. Mochida, K. Kouno, Y. Hayata, M. Nakayama, T. Ono, H. Suwa, R. Yasuhara, K. Katayama, T. Mikawa, Y. Gohou, in 2018 IEEE Symp. on VLSI Technology, Honolulu, HI, USA 2018, pp. 175–176.
17. C. Mackin, H. Tsai, S. Ambrogio, P. Narayanan, A. Chen, C. G. Burr, Adv. Electron. Mater. 2019, 5, 1900026.
18. M. Hu, C. E. Graves, C. Li, Y. Li, N. Ge, E. Montgomery, N. Davila, H. Jiang, R. S. Williams, J. J. Yang, Q. Xia, J. P. Strachan, Adv. Mater. 2018, 30, 1705914.
19. A. Valentin, F. Rummens, E. Vianello, T. Mesquida, C. Lacat-Mathieu de Boissac, C. Reita, in International Electron Device Meeting Device, San Francisco, CA, USA 2019.
20. D. Querlioz, O. Bichler, A. F. Vincent, C. Gamrat, Proc. IEEE 2015, 103, 1398.
21. I. Boybat, M. Gallo, S. R. Nandakumar, T. Moraitis, T. Parnell, T. Tuma, B. Rajendran, Y. Leblebici, A. Sebastian, E. Eleftheriou, Nat. Commun. 2017, 9, 2514.
22. M. Bocquet, T. Hirztlin, J.-O. Klein, E. Nowak, E. Vianello, J.-M. Portal, D. Querlioz, in 2018 IEEE International Electron Devices Meeting (IEDM), IEEE, Piscataway, NJ 2018, pp. 20–26.
23. T. Hirztlin, M. Bocquet, B. Penkovsky, J.-O. Klein, E. Nowak, E. Vianello, J.-M. Portal, D. Querlioz, Front. Neurosci. 2020, 13, 1383.
24. T. Daligay, M. Payvand, F. Moro, D. R. B. Ly, F. Peyab-Peyroula, J. Casas, G. Indiveri, E. Vianello, APL Mater. 2019, 7, 083125.
25. T. Daligay, N. Castellani, C. Turck, K.-E. Harabi, D. Querlioz, E. Vianello, Nat. Electron. 2021, 4, 151.
26. J. S. Friedman, L. E. Calvert, P. Bessière, J. Droulez, D. Querlioz, IEEE Trans. Circuits Syst. I: Regular Papers 2016, 63, 895.
27. R. Faria, K. Y. Camsari, S. Datta, Front. Neurosci. 2018, 8, 045101.
28. C. S. Thakur, S. Afshar, R. M. Wang, T. J. Hamilton, J. Tapson, A. Van Schaik, Front. Neurosci. 2016, 10, 104.
29. P. Mroszczyk, P. Dudek, in 2014 IEEE International Symposium on Circuits and Systems (ISCAS), IEEE, Piscataway, NJ 2014, pp. 1576–1579.
30. Z. Ghahramani, Nature 2015, 521, 452.
31. R. M. Neal, Bayesian Learning for Neural Networks, Vol. 118, Springer Science & Business Media, Cham 2012.
32. A. Grossi, E. Nowak, C. Zambelli, C. Pellissier, S. Bernasconi, G. Cibrario, K. E. Hajjarn, R. Crochemore, J. F. Nodin, P. Olivo,
[33] D. Ielmini, IEEE Trans. Electron Devices 2011, 58, 4309.
[34] T. K. Moon, IEEE Signal Process. Mag. 1996, 13, 47.
[35] A. Grossi, E. Vianello, M. M. Sabry, M. Barlas, L. Grenouillet, J. Coignus, E. Beigne, T. Wu, B. Q. Le, M. K. Wootters, C. Zambelli, E. Nowak, S. Mitra, IEEE Trans. Electron Devices 2019, 66, 1281.
[36] P. Hsu, H. Robbins, Proc. Natl. Acad. Sci. USA 1947, 33, 25.
[37] A. Kucukelbir, D. Tran, R. Ranganath, A. Gelman, D. M. Blei, J. Mach. Learn. Res. 2017, 18, 430.
[38] M. Hoffman, A. Gelman, J. Mach. Learn. Res. 2014, 15, 1351.
[39] T. Wiecki, M. Kochurov, Bayesian Deep Learning, 2016, https://twiecki.io/blog/2016/06/01/bayesian-deep-learning.
[40] D. D. Fan, J. Nguyen, R. Thakker, N. Alatur, A.-A. Agha-Mohammadi, E. A. Theodorou, 2020 IEEE International Conference on Robotics and Automation (ICRA), Paris, France 2019, pp. 4093–4099.
[41] B. J. Maron, W.-K. Shen, M. S. Link, A. E. Epstein, A. K. Almquist, J. P. Daubert, G. H. Bardy, S. Favale, R. F. Rea, G. Boriani, N. A. Estes, P. Spirito, New Engl. J. Med. 2000, 342, 365.
[42] J. P. Daubert, W. Zareba, D. S. Cannom, S. McNitt, S. Z. Rosero, P. Wang, C. Schuger, J. S. Steinberg, S. L. Higgins, D. J. Wilber, H. Klein, M. L. Andrews, W. J. Hall, A. J. Moss, J. Am. College Cardiol. 2008, 51, 1357.