Spiking neural networks for nonlinear regression

Journal Article

Author(s):
Henkes, Alexander; Eshraghian, Jason K.; Wessels, Henning

Publication date:
2024-05-01

Permanent link:
https://doi.org/10.3929/ethz-b-000673051

Rights / license:
Creative Commons Attribution 4.0 International

Originally published in:
Royal Society Open Science 11(5), https://doi.org/10.1098/rsos.231606

This page was generated automatically upon download from the ETH Zurich Research Collection. For more information, please consult the Terms of use.
Spiking neural networks for nonlinear regression

Alexander Henkes\textsuperscript{1,2}, Jason K. Eshraghian\textsuperscript{3} and Henning Wessels\textsuperscript{2}

\textsuperscript{1}Computational Mechanics Group, ETH Zurich, Zurich, Switzerland
\textsuperscript{2}Division Data-Driven Modeling of Mechanical Systems, Technical University Braunschweig, Braunschweig, Germany
\textsuperscript{3}Department of Electrical and Computer Engineering, University of California, Santa Cruz, CA, USA

Spiking neural networks (SNN), also often referred to as the third generation of neural networks, carry the potential for a massive reduction in memory and energy consumption over traditional, second-generation neural networks. Inspired by the undisputed efficiency of the human brain, they introduce temporal and neuronal sparsity, which can be exploited by next-generation neuromorphic hardware. Energy efficiency plays a crucial role in many engineering applications, for instance, in structural health monitoring. Machine learning in engineering contexts, especially in data-driven mechanics, focuses on regression. While regression with SNN has already been discussed in a variety of publications, in this contribution, we provide a novel formulation for its accuracy and energy efficiency. In particular, a network topology for decoding binary spike trains to real numbers is introduced, using the membrane potential of spiking neurons. Several different spiking neural architectures, ranging from simple spiking feed-forward to complex spiking long short-term memory neural networks, are derived. Since the proposed architectures do not contain any dense layers, they exploit the full potential of SNN in terms of energy efficiency. At the same time, the accuracy of the proposed SNN architectures is demonstrated by numerical examples, namely different material models. Linear and nonlinear, as well as history-dependent material models, are examined. While this contribution focuses on mechanical examples, the interested reader may regress any custom function by adapting the published source code.

© 2024 The Authors. Published by the Royal Society under the terms of the Creative Commons Attribution License http://creativecommons.org/licenses/by/4.0/, which permits unrestricted use, provided the original author and source are credited.
1. Introduction

In recent years, artificial neural networks (ANN) have gained much attention in the computational engineering sciences and applied mathematics due to their flexibility and universal approximation capabilities, both for functions [1,2] and operators [3]. Their outstanding but surprising generalizability capabilities are yet to be understood [4]. Their advantages have been used in a variety of applications, including fluid dynamics [5–9], solid mechanics [10–13], micromechanics [14–17], material parameter identification [18–22], constitutive modelling [23–26], fracture mechanics [27,28], microstructure generation [29–31], contact problems [32–35], heat transfer [36–39], uncertainty quantification [40–43], among numerous others. Refer to [44–46] for review publications.

Despite the success of ANNs, several problems arise alongside their use, such as the need for high-frequency memory access, which leads to high computational power demand [47,48]. This results in huge costs for training and often makes it preferable to run inference in remote servers during deployment. In general, ANNs are most often trained on graphics processing units (GPUs), whose energy consumption is problematic in embedded systems (e.g. sensor devices) as is required in automotive and aerospace applications [49] or structural health monitoring (SHM). Furthermore, high latency during prediction time can arise where acceleration or parallelization is not available. Figure 1 illustrates a potential use case of neuromorphic hardware in SHM. Using concepts from data-driven mechanics, sensor data collected during service life may be interpreted with the aid of high-dimensional (computational) physical models, which so far have only been used to design infrastructures or products. Thanks to active research, the computational efficiency of such advanced monitoring concepts is rapidly increasing (e.g. [50], for a recent contribution). A bottleneck for practical online applications, however, remains the aforementioned hardware constraints, namely, energy demand and latency. Regression using spiking neural networks (SNN) on neuromorphic hardware may contribute to overcome persisting limitations.

Originally motivated by the human brain, today’s traditional ANN architectures are an oversimplification of biology, relying on dense matrix multiplication. From a numerical and computational hardware point of view, dense matrix multiplication is often suboptimal. Sparsity is thought to be favourable as it reduces dependence on memory access and data communication [51]. In contrast, the human brain is much more efficient, where neurons are considered to be sparsely activated [52]. This stems from the fact that the brain uses sparse electronic signals for information transmission instead of dense activations. This leads to remarkable capabilities by using only approximately 10–20 W of energy. One attempt to overcome these drawbacks of ANNs is to introduce the information transmission mechanism of biological neurons into network architectures. These networks are called SNN due to the electronic impulses or spikes used for communication between neurons [53]. This leads to sparse activations, which can be efficiently exploited by neuromorphic hardware, such as Loihi [54], SpiNNaker [55] and TrueNorth [56]. It has been shown that these specialized hardware chips can reduce the energy consumption of neural network-based processes by factors of up to ×1000 [54,57–60]. Apart from their energy efficiency in prediction, recent attempts to increase the training efficiency of SNN can be found in [61–64].

What was classically in the domain of neuroscientists recently has been investigated in the context of deep learning, for example, the adoption of SNNs to supervised learning as popularized with traditional ANNs in frameworks, such as TensorFlow [65] and PyTorch [66], resulting in similar frameworks for spiking deep learning like snnTorch [67]. Some applications of spiking deep learning include image processing using a spiking ResNet [68] and temporal data processing using spiking long short-term memory neural network (SLSTM) variants [69,70]. A combination of spiking convolutional neural networks and LSTMs was proposed in Yang et al. [71]. SNNs have been used for image segmentation [72] and localization [73,74].

To the best of the authors’ knowledge, the full potential of energy-efficient regression modelling using SNNs has up to now not yet been fully exploited. In Iannella and Back [75], an architecture using inter-spike interval temporal encoding has been proposed, where the functions learned were limited to piecewise constant functions. In Gehrig et al. [76], a SNN was used for the regression of angular velocities of a rotating event camera. Building on these results, Rançon et al. [77] proposed a SNN for depth reconstruction. Both works rely on fully connected decoders for real-valued output, which increases the energy demand compared with an architecture relying only on SNN predictions. This is due to the increased number of dense matrix multiplications in fully connected layers. In Kahana et al. [78], a DeepONet [79] using SNN was proposed, which used a floating point decoding scheme to
regress simple one-dimensional functions. This decoding scheme led to non-smooth and staircase-like predictions. In Shrestha and Orchard [80], gradient descent was applied to learn spike times. Here, as in the present contribution, the membrane potential is used, but to tweak the occurrence of the first spikes in the SNN. In Eshraghian et al. [81], classification problems using the membrane potential in the context of memristor-based hardware were explored, but no regression tasks. Recently, a SNN has been proposed in a computational mechanics context in Tandale and Stoffel [82], which builds up on our approach developed in the preprint [83]. However, again fully connected layers were added, which counteract the energy efficiency.

The focus of the present work lies on network architectures that come along without any fully connected layers and are therefore optimally suited for neuromorphic hardware, which is specifically designed for SNNs. The proposed regression approach works without fully connected dense encoders/decoders by using only spiking layers. The output of the proposed networks is real-valued, continuous functions and can represent highly nonlinear, history-dependent data accurately.

As regression problems are omnipresent in engineering sciences, a flexible and broadly applicable framework would enable SNNs to be used in a variety of engineering applications and further unfold the potential of neuromorphic hardware. To this end, the present study aims towards the following key contributions:

— **Introduction of spiking neural networks.** Concise introduction of this emerging technique. Open source benchmark code for the research community, fostering further developments of SNN in mechanics and applied mathematics (see also Data accessibility section).

— **History-dependent regression framework.** We demonstrate that SNNs can model systems that exhibit hysteresis, thus history-dependent behaviour. We present a flexible framework to use SNNs in complex regression tasks, demonstrated by means of a nonlinear material model, namely isotropic hardening plasticity.

— **Efficiency, sparsity and latency.** We benchmark our SNN for neuromorphic hardware in terms of memory and energy consumption as compared with non-spiking equivalent networks, demonstrating that our approach is much more efficient with respect to memory and power consumption, making neural networks more sustainable. This is enabled by a membrane potential-based decoding scheme, which does not rely on fully connected decoders. While in our study an
emulator is used, deployment on neuromorphic hardware allows highly efficient usage in embedded environments.

The present work intends to introduce this important novel technique to the community of computational mechanics and applied mathematics. To concentrate on the novelties and to keep the presentation concise, we restrict ourselves to one-dimensional regression problems. However, the framework is not restricted to single-variable regression and is easily applicable to a multi-variable regression.

The remainder of this article is structured as follows. In §2, the basic notations of SNNs are derived from traditional ANNs. A simple spiking counterpart to the classical densely connected feed-forward neural network is introduced. After that, our regression SNN topology is proposed. Starting with a linear regression example (linear elasticity), we point out the problems arising in SNN regression. This basic architecture is extended towards recurrent feedback loops in §3. The ability of these recurrent SNNs is showcased on a nonlinear material model. To process history-dependent regression tasks with dependencies over a large number of time steps, a spiking LSTM is introduced in §4. An application to a history-dependent plasticity model shows that SNNs can achieve similar accuracy as their traditional counterparts while being much more efficient. The paper closes with a conclusion and an outlook towards future research directions in §5. For the code accompanying this manuscript, see the Data accessibility section at the end of this manuscript.

2. Spiking neural networks for regression

SNNs are considered to be the third generation of neural networks. While the first generation was restricted to shallow networks, the second generation is characterized by deep architectures. A broad use of second-generation neural networks has been enabled by the availability of automatic differentiation and software frameworks such as TensorFlow [65]. To introduce SNN, we compare them with their well-known second-generation counterparts.

2.1. Second-generation neural networks

An ANN is a parametrized, nonlinear function composition. The universal function approximation theorem [1] states that arbitrary Borel measurable functions can be approximated with ANNs. There are several different architectures for ANNs, for example, feed-forward, recurrent or convolutional networks, which can be found in standard references such as [84–88]. Following Hauser [89], most ANN formulations can be unified. An ANN $\mathcal{N}$, more precisely, a densely connected feed-forward neural network, is a function from an input space $\mathbb{R}^{d_x}$ to an output space $\mathbb{R}^{d_y}$, defined by a composition of nonlinear functions $h^{(l)}$, such that

\[
\mathcal{N}: \mathbb{R}^{d_x} \rightarrow \mathbb{R}^{d_y}, \quad \mathbf{x} \mapsto \mathcal{N}(\mathbf{x}) = h^{(L)} \circ \cdots \circ h^{(1)} = \mathbf{y},
\]

\[l = 1, \ldots, L. \tag{2.1}\]

Here, $\mathbf{x}$ denotes an input vector of dimension $d_x$ and $\mathbf{y}$ an output vector of dimension $d_y$. The nonlinear functions $h^{(l)}$ are called layers and define an $l$-fold composition, mapping input vectors to output vectors. Consequently, the first layer $h^{(1)}$ is defined as the input layer and the last layer $h^{(L)}$ as the output layer, such that

\[
h^{(1)} = \mathbf{x} \in \mathbb{R}^{d_x}, \quad h^{(L)} = \mathbf{y} \in \mathbb{R}^{d_y}. \tag{2.2}\]

The layers $h^{(l)}$ between the input and output layers, called hidden layers, are defined as

\[
\begin{align*}
    h^{(l)} &= \{h^{(l)}_\eta, \eta = 1, \ldots, n_u\}, \\
    h^{(l)}_\eta &= \phi^{(l)}(W^{(l)}_\eta \cdot h^{(l-1)}).
\end{align*} \tag{2.3}\]

where $h^{(l)}_\eta$ is the $n$-th neural unit of the $l$-th layer $h^{(l)}$, $n_u$ denotes the total number of neural units per layer, $W^{(l)}_\eta$ is the weight vector of the $n$-th neural unit in the $l$-th layer $h^{(l)}$ and $h^{(l-1)}$ is the output of the
2.2. Spiking neural networks

A SNN can be seen as a history-dependent ANN, which introduces memory effects by means of biologically inspired processes. Standard works in theoretical neuroscience include the studies of Dayan and Abbott [90], Izhikevich [91] and Gerstner et al. [92]. Several overviews of SNNs with respect to deep learning can be found in [93–95].

To this end, the activation function $\phi^{(l)}$ in equation (2.5) can be formulated as

$$\phi^{(l)}_{\text{spk},t} = \begin{cases} 1, & U_{\eta, t}^{(l)} \geq U_{\text{thr}, \eta}^{(l)} \\ 0, & U_{\eta, t}^{(l)} < U_{\text{thr}, \eta}^{(l)} \end{cases}$$

with

$$U_{\eta, t}^{(l)} = \beta_{\eta}^{(l)} U_{\eta, t-1}^{(l)} + W_{\eta}^{(l)} h_{t-1}^{(l-1)} - \phi_{\text{spk}, t-1}^{(l)} U_{\text{thr}, \eta}^{(l)}$$

where $U_{\eta, t}^{(l)}$ is the membrane potential of the $\eta$-th neural unit at time $t$, $U_{\text{thr}, \eta}^{(l)}$ denotes the membrane threshold, $\beta_{\eta}^{(l)}$ is the membrane potential decay rate and $W_{\eta}^{(l)} h_{t-1}^{(l-1)}$ is the standard ANN weight multiplied with the preceding layer of the current time step, respectively, see equation (2.5). Basically, the SNN activation restricts the neural unit to output discrete pulses ($\phi_{\text{spk}} = 1$) if the membrane potential exceeds the threshold. This can be understood as processing each discrete-time slice of the input vector of the preceding layer $h_{t-1}^{(l-1)} \rightarrow h_{t-1}^{(l-1)} \rightarrow \ldots \rightarrow h_{t-1}^{(l-1)}$ sequentially, where the weights $W_{\eta}^{(l)}$ are shared over all time steps. At this stage, the formulation in equation (2.5) is purely notational, as there is no connection of the weights through different time steps.
threshold is reached by the time-evolving membrane potential, or to remains silent ($\phi_{spk} = 0$). These pulses are called spikes, see Figure 3. The last summand in equation (2.6), $-\phi_{spk,t-1} U_{thr,\eta}$ is called the reset mechanism and resets the membrane potential by the threshold potential once a spike is emitted. The membrane threshold and membrane potential decay rate can be optimized during training, such that the optimization parameters of a SNN are

$$\theta_{SNN} = \{W_{\eta}^{(l)}, \beta_{\eta}^{(l)}, U_{thr,\eta}^{(l)}\}.$$ (2.8)

The SNN formulation in equation (2.7) is called the leaky integrate and fire (LIF) neuron model, and is one of the most widely used models in spike-based deep learning. It can be seen as the baseline SNN and plays a similar role as densely connected feed-forward ANN in classical deep learning. Equation (2.7) can also be interpreted as the explicit forward Euler solution of an ordinary differential equation, describing the time variation of the membrane potential, see Eshraghian et al. [67] for details.

The main difference between SNNs and classical ANNs lies in the way information is processed and propagated through the network from neuron to neuron. In standard ANNs, inputs, hidden layers and output vectors are handled via dense matrices. In SNN, sparsity is introduced by using spikes, which are single events expressed via a Dirac delta function or a discrete pulse in continuous or discrete settings, respectively. A group of spikes over time is called a spike train $i_t = i, t = 0, \ldots, n_t$. To this end, a spiking neuron is subjected to a spike train over a time interval, consisting of spikes (1) or zero input (0). The membrane potential $U_{\eta,t}^{(l)}$ is modulated with incoming spikes $i_t$. In the absence of input spikes, the membrane voltage decays over time due to the membrane decay rate $\beta_{\eta}^{(l)}$. The absence of spikes introduces sparsity because, in every time step, the neural unit output is constrained to either 0 or 1. This fact can be exploited on neuromorphic hardware, where memory and synaptic weights need only be accessed if a spike is apparent. Otherwise, no information is transmitted. In contrast, conventional ANNs do not leverage sparsely activated neurons, and most deep learning accelerators, such as GPUs or tensor processing units (TPUs), are correspondingly not optimized for it.

### 2.3. Spiking neural networks training

Unfortunately, the spiking activation $\phi_{spk,t}^{(l)}$ in equation (2.6) is non-differentiable. To use the backpropagation algorithm from standard ANNs, the activation is replaced using a surrogate gradient during the backward pass. Several different formulations have been proposed (e.g. [51,96,97]). In this work, the arcus tangent surrogate activation from Fang et al. [98] is used,

$$\phi_{surr}(x) = \frac{1}{\pi} \arctan(\pi x), \quad \phi_{surr}'(x) = \frac{1}{1 + (\pi x)^2}.$$ (2.9)
for some input $x$. The surrogate $\phi_{\text{surr}}(x)$ is continuously differentiable and preserves the gradient dynamics of the network. Thus, for training using backpropagation and its variants, $\phi_{\text{surr}}$ is employed. Illustrations can be found in figures 4 and 5.

2.4. Network topology

The key question for using SNN in regression is how to transform real input values into binary spikes and binary spike information at the output layer back to real numbers. The former task is called spike encoding, whereas the latter is called spike decoding. Popular forms of encoding include rate encoding, latency encoding, delta modulation, and second-moment estimates. The parameters are set as follows: learning rate $\alpha = 1 \times 10^{-3}$, exponential decay rates for the first- and second-moment estimates $\beta_1 = 0.9$ and $\beta_2 = 0.999$, respectively, weight decay $\lambda = 0.01$. The training was carried out on a Nvidia GeForce RTX 3090 GPU using snnTorch [67] and PyTorch [66]. In this work, the mean relative error $\mathcal{E}$ is used, which is defined as

$$\mathcal{E} = \frac{1}{n} \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right|,$$

where $n$ is the number of data points.

A graphical interpretation is given in figure 7. For all the following numerical examples, the AdamW optimizer from Loshchilov and Hutter [99] is used. The parameters are set as follows: learning rate $\alpha = 1 \times 10^{-3}$, exponential decay rates for the first- and second-moment estimates $\beta_1 = 0.9$ and $\beta_2 = 0.999$, respectively, weight decay $\lambda = 0.01$. The training was carried out on a Nvidia GeForce RTX 3090 GPU using snnTorch [67] and PyTorch [66]. In this work, the mean relative error $\mathcal{E}$ is used, which is defined as

$$\mathcal{E} = \frac{1}{n} \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right|,$$
\[ \mathcal{E} \cdot (\cdot) = \sum_{i=1}^{n_s} |\mathbf{x}_i - \mathbf{x}_i^\circ|^2, \]

for some input \( \cdot \) and baseline \( \mathbf{x}_i^\circ \). If the error is reported for all time steps, \( \cdot \) is a vector containing the values of all time steps. If the error is reported for the last time step, \( \cdot \) is equal to the last component of the corresponding history-dependent vector.

### 2.5. Numerical experiment: linear regression

We first study the performance of the proposed LIF topology in a simple linear regression problem. To this end, the general model described in §2.1 with LIF defined in equation (2.7) is used, resulting in the following network topology.

\[ S(\varepsilon) = h_{\text{pop}} \circ h_{\text{dec}}^{\text{LIF}} \circ h_{\text{LIF}} \circ h_{\text{const}}(\varepsilon) = \sigma, \]

To begin with, a simple linear elastic material model with strains in the range of \( \varepsilon = [0, 0.001] \) and fixed Young’s modulus \( E = 2.1 \times 10^5 \text{ MPa} \) is considered, such that the resulting stress \( \sigma \) is
The training data consists of strain input, uniformly sampled in the interval $\varepsilon = [0, 0.001]$, and stress output calculated according to equation (2.16). Three datasets are generated from equation (2.16), namely, training set, validation set and test set consisting each of $n_{\text{train}} = n_{\text{val}} = n_{\text{test}} = 1024$ samples. All three datasets are standardized using the mean and standard deviation from the training set. The batch size is chosen as $n_{\text{batch}} = 1024$. The number of neurons $n_u$ is chosen as $n_u = 128$ and is kept constant over all layers. The training is carried out for $2 \times 10^3$ epochs. The model performing best on the validation set is chosen for subsequent evaluations. The mean relative error accumulated over all time steps and the mean relative error of the last time step with respect to the test set are reported.

Although the linear regression problem is a time-independent problem, SNN has an inherent time dependency. Instead of time, the SNN is thus trained on a sequence of strain steps $\Delta \varepsilon$. We investigate...
the effect of the number of strain (time) steps on the prediction accuracy, with results illustrated in figure 8.

It can be seen that the mean relative error is lowest for $d_t = 5$ strain steps. For $d_t = 2$, the error is larger. This could be caused by a lack of a sufficient number of strain steps for the neuron dynamics to effectively be calculated. It can be understood as a failure due to too large strain steps in the explicit stepping scheme in equation (2.7). Clearly, the highest error can be observed for $d_t = 100$ strain steps. In contrast, as depicted in figure 9, the error at the last strain step is lowest for $d_t = 100$ strain steps. To illustrate the cause, the prediction of the network for two different samples, one for $d_t = 5$ and one for $d_t = 100$ strain steps are shown in figures 10 and 11, respectively. While good agreement on the endpoints is apparent, fluctuation during the rest of the strain steps causes the error to rise. Seemingly, the LIF has difficulties regressing a large number of strain steps. This could be caused by the lack of recurrent connections in the LIF formulation from equation (2.15), where history dependency is only weakly included in the form of the membrane potential. Therefore, recurrent LIFs will be introduced in §3.

Remark: The seemingly simple linear regression task provides a challenge for SNN, as effectively an ordinary differential equation has to be fitted to a linear function while relying on binary information transmission and inexact gradients.

3. Nonlinear regression using recurrent leaky integrate and fire

In order to counter the problems of vanishing information for a large number of strain (time) steps encountered in the preceding section, a recurrent SNN architecture is proposed (§3.1). Its performance is demonstrated by means of a numerical example in §3.2.

3.1. Recurrent leaky integrate and fire

The standard LIF is a feed-forward neuron, such that information is flowing unidirectionally in the form of spikes. By adding a feedback loop, a recurrent LIF (RLIF) can be formulated, which builds on the standard recurrent neural network (RNN) formulation. This enables the network to use relationships along several time steps for the prediction of the current time step. It was shown in Pascanu et al. [100] that recurrent loops can retain information for a relatively longer number of time steps when compared with their non-recurrent counterparts.

Here, the formulation of the hidden layer in equation (2.5) includes additional recurrent weights $V^{(l)}_\eta$, such that

$$ h^{(l)}_{\eta,t} = \varphi^{(l)}(W^{(l)}_\eta h^{(l-1)}_{\eta} + V^{(l)}_\eta h^{(l-1)}_{\eta-1}). $$

In this RNN, the influence of the preceding time step is explicitly included by means of additional recurrent weights $V^{(l)}_\eta$. The resulting set of trainable parameters reads

$$ \theta_{\text{RNN}} = \{ W^{(l)}_\eta, V^{(l)}_\eta \}. $$

The RNN formulation can be included in the LIF formulation from equation (2.7) to obtain an RLIF, such that

$$ U^{(l)}_{\eta,t} = \rho^{(l)} U_{\eta,t-1}^{(l)} + W^{(l)}_\eta h^{(l-1)}_{\eta-1} $$

$$ + V^{(l)}_\eta h^{(l-1)}_{\eta-1} - \varphi^{(l)} U^{(l)}_{\text{thr}, \eta}, $$

where $U_{\eta,t}^{(l)}$ is again the membrane potential of the $\eta$th neural unit at time $t$, $U^{(l)}_{\text{thr}, \eta}$ denotes the membrane threshold, $\rho^{(l)}$ is the membrane potential decay rate and $W^{(l)}_\eta h^{(l-1)}_{\eta-1}$ is the standard ANN weight multiplied with the preceding layer at the current time step. Additionally, $V^{(l)}_\eta$ denotes the recurrent weights from equation (3.1). This leads to the following set of trainable parameters

$$ \theta_{\text{RLIF}} = \{ W^{(l)}_\eta, V^{(l)}_\eta, \rho^{(l)}, U^{(l)}_{\text{thr}, \eta} \}. $$
**Figure 8.** Elasticity—error of all strain (time) steps. Mean relative error for all strain steps with respect to the different total number of strain steps. The error is rising for a larger number of strain steps.

**Figure 9.** Elasticity—error at last strain (time) step. Mean relative error for the last strain step with respect to the different total number of strain steps. The error is converging for a larger number of strain steps.

**Figure 10.** Elasticity—prediction in five strain (time) steps. Prediction of the LIF from equation (2.15) for $d_t = 5$. Deviations from the true solution can be observed in the middle part.
3.2. Numerical experiment: Ramberg–Osgood

The performance of the RLIF is investigated in nonlinear function regression. As a first test case, the nonlinear Ramberg–Osgood power law for modelling history-independent plasticity is considered, in which stress $\sigma$ and strain $\varepsilon$ are related via

$$\varepsilon = \frac{\sigma}{E} + 0.002 \left( \frac{\sigma}{\sigma_Y} \right)^n. \quad (3.5)$$

Here, $\varepsilon$ is the infinitesimal, one-dimensional elastic strain, $\sigma$ denotes the one-dimensional Cauchy stress, $E$ is Young’s modulus, $n$ is a material constant describing the hardening behaviour of plastic deformation and $\sigma_Y$ is the yield strength of the material. Note that this plasticity model is only suited for a single loading direction and does not incorporate accumulation of plastic strain.

To this end, the general model described in §2.1 using RLIF defined in equation (3.3) is used. For a given strain history $\varepsilon_t$, the stress response for different values of yield stress $\sigma_Y$ is predicted with the following parametric architecture

$$\sigma_Y = h_t(\sigma_Y) = h_t^{\text{const}}(\sigma_Y) = \sigma_t. \quad (3.6)$$

The training data consists of yield strength $\sigma_Y$ as input for fixed strains in the interval $\varepsilon = [0, 0.01]$ for $d_t = 20$ strain steps. The yield strength is uniformly sampled in the interval $\sigma_Y = [100, 500]$ MPa, and the stress output is calculated according to equation (3.5). The Young’s modulus is chosen as $E = 2.1 \times 10^5$ MPa and $n = 10$. Three datasets are generated, namely, training set, validation and test set with $n_{\text{train}} = n_{\text{val}} = n_{\text{test}} = 1024$ samples, respectively. All three sets are standardized using the mean and standard deviation from the training set. The batch size is chosen as $n_{\text{batch}} = 1024$. The number of neurons $n_u$ is chosen as $n_u = 128$ and is kept constant over all layers. The training is carried out for $5 \times 10^3$ epochs. The model performing best on the validation set is chosen for subsequent evaluations. The mean relative error and the mean relative error of the last strain step with respect to the test set are reported.

In figure 12, different stress–strain curves are depicted for different yield strength values, obtained by solving equation (3.5) for the stress $\sigma$ with a classical Newton–Raphson method. The results of five different samples, randomly chosen from the $n_{\text{test}} = 1024$ test samples, can be seen in figure 13. For the test set, a mean relative error for all strain steps of $8.7934 \times 10^{-2}$ and a mean relative error for the last strain step of $8.0200 \times 10^{-2}$ are obtained. The predictions on these five samples are more accurate than would be suspected from the mean relative error. The cause can be found in figure 14, where the mean relative error for all strain steps is plotted for every sample of the test set. It can be observed that a small number of samples has a much higher error than the rest, which impacts the error measure.

![Figure 11](Figure 11. Elasticity—prediction in 100 strain (time) steps. Prediction of the LIF from equation (2.15) for $d_t = 100$. Fluctuations around the true solution can be observed.)
It is caused by the purely data-driven nature of the experiment and can be tackled with approaches introduced in, for example, [101–103].

Nevertheless, the RLIF is able to regress on the varying yield strength \( \sigma_Y \) and can predict the resulting nonlinear stress–strain behaviour, as can be seen in the predictions (figure 13). Deviations can be observed around the yield point as well as the endpoints of the curves. To be able to take into account long-term history-dependent behaviour, the RLIF formulation will be expanded towards the incorporation of explicit long-term memory in the next section, where a more complex plasticity model is investigated.

4. History-dependent regression using spiking long short-term memory network

To extend the limited memory of the RNN in §3, a SLSTM (§4.1) is proposed. Previously, SLSTM has been considered for classification problems [71]. Herein, a novel regression approach with a population decoding layer is discussed by means of a history-dependent plasticity model (§4.2).

4.1. Spiking long short-term memory network

A SLSTM is the spiking version of the standard LSTM [104], where the latter is defined as

\[
h^{(t)}_{t_l} = o^{(t)}_{t_l} \odot \phi_{\text{tanh}}(c^{(t)}_{t_l-1}),
\]

(4.1)

with

\[
o^{(t)}_{t_l} = \phi_{\text{sigmoid}}(W^{(t)}_{i_l} h^{(t-1)}_{t_l} + V^{(t)}_{i_l} h^{(t-1)}_{t_l-1}),
\]

\[
c^{(t)}_{t_l} = f^{(t)}_{t_l} \odot c^{(t)}_{t_l-1} + i^{(t)}_{t_l} \odot \tilde{c}^{(t)}_{t_l},
\]

\[
f^{(t)}_{t_l} = \phi_{\text{sigmoid}}(W^{(t)}_{f_l} h^{(t-1)}_{t_l} + V^{(t)}_{f_l} h^{(t-1)}_{t_l-1}),
\]

\[
h^{(t)}_{t_l} = \phi_{\text{sigmoid}}(W^{(t)}_{o_l} h^{(t-1)}_{t_l} + V^{(t)}_{o_l} h^{(t-1)}_{t_l-1}),
\]

\[
\tilde{c}^{(t)}_{t_l} = \phi_{\text{tanh}}(W^{(t)}_{c_l} h^{(t-1)}_{t_l} + V^{(t)}_{c_l} h^{(t-1)}_{t_l-1}),
\]

where \( f \) denotes the forget gate with sigmoid activation \( \phi_{\text{sigmoid}} \) or tangent hyperbolicus activation \( \phi_{\text{tanh}} \) and corresponding weights \( W_f, V_f \) with absorbed biases. The same nomenclature holds for the input gate \( i \), the output gate \( o \), the cell input \( \tilde{c} \), and the cell state \( c \) with their respective activations and weights. The new cell state \( c \) and the output of the LSTM \( h \) are formed using the Hadamard or point-wise product \( \odot \). The parameters of the LSTM are its weights, such that

\[
\theta_{\text{LSTM}} = \{ W^{(t)}_{f_l}, W^{(t)}_{i_l}, W^{(t)}_{o_l}, W^{(t)}_{c_l}, V^{(t)}_{f_l}, V^{(t)}_{i_l}, V^{(t)}_{o_l}, V^{(t)}_{c_l} \}.
\]

(4.2)

For detailed derivations and explanations of standard LSTM, see, for example, [85,86]. The SLSTM can be obtained from the LSTM by using spike activations within the LSTM formulation from equation (4.1), such that

\[
h^{(t)}_{t_l} = o^{(t)}_{t_l} \odot \phi_{\text{tanh}}(c^{(t)}_{t_l-1}) \odot \phi_{\text{spk}, t_l-1} U^{(t)}_{\text{thr}, t_l},
\]

(4.3)

where the output \( h^{(t)}_{t_l} \) is used to determine if a spike is produced

\[
\phi_{\text{spk}, t_l} = \begin{cases} 1, & h^{(t)}_{t_l} \geq U^{(t)}_{\text{thr}, t_l} \\ 0, & h^{(t)}_{t_l} < U^{(t)}_{\text{thr}, t_l}. \end{cases}
\]

(4.4)

In other words, the output of \( h^{(t)}_{t_l} \) can be interpreted as the membrane potential of the SLSTM, such that \( h^{(t)}_{t_l} = U^{(t)}_{\text{S}_{t_l}} \). A decay parameter \( \beta \) is not used in this formulation. Rather than using decay to remove information from the cell state \( c^{(t)}_{t_l} \), this is achieved by carefully regulated gates. The corresponding optimization parameters of the SLSTM are
\[
\theta_{\text{SLSTM}} = \{ W_f, \eta_l, W_i, \eta_l, W_o, \eta_l, W_c, \eta_l, V_f, \eta_l, V_i, \eta_l, V_o, \eta_l, V_c, \eta_l, U_{\text{thr}}, \eta_l \}.
\]

Basically, the cell state \( c^{(l)}_{\eta, t} \) acts as long-term memory, just like in the standard LSTM formulation. The communication between layers is handled via spike trains that depend on the membrane potential \( h^{(l)}_{\eta, t} = U^{(l)}_{\eta, t} \) in equation (4.3) and the activation function \( \phi^{(l)}_{\text{spk}, t} \) from equation (4.4).

### 4.2. Numerical experiment: Isotropic hardening using spiking long short-term memory network

The following numerical experiments aim to investigate the performance of the proposed SLSTM on nonlinear, history-dependent problems. Therefore, a one-dimensional plasticity model with isotropic hardening is investigated. Following Simo and Hughes [105], the model is defined by
1. \( \varepsilon = \varepsilon_{el} + \varepsilon_{pl} \)
2. \( \sigma = E(\varepsilon - \varepsilon_{pl}) \)
3. \( \dot{\varepsilon}_{pl} = \gamma \text{sign}(\sigma), \quad \dot{\alpha} = \gamma \)
4. \( f(\sigma, \alpha) = |\sigma| - (\sigma_Y + K\alpha) \leq 0 \)
5. \( \gamma \geq 0, \quad f(\sigma, \alpha) \leq 0, \quad \gamma f(\sigma, \alpha) = 0 \)
6. \( \gamma f(\sigma, \alpha) = 0, \quad \text{if } f(\sigma, \alpha) = 0 \) \hspace{1cm} (4.6)

where

1. is the additive elastoplastic split of the small-strain tensor \( \varepsilon \) into a purely elastic part \( \varepsilon_{el} \) and a purely plastic part \( \varepsilon_{pl} \);
2. denotes the elastic stress–strain relationship for the Cauchy stress tensor \( \sigma \) and elastic modulus \( E \);
3. describes the flow rule and isotropic hardening law with consistency parameter \( \gamma \) and equivalent plastic strain \( \alpha \);
4. gives the yield condition \( f(\sigma, \alpha) \) with hardening modulus \( K \);
5. denotes the Kuhn–Tucker complementarity conditions; and
6. describes the consistency condition.

In figure 15, different stress–strain paths are shown for varying strains. Especially long-time dependencies are of interest. To this end, the predictive capabilities of the SNN are investigated for inference over \( d_t = 100 \) strain steps, where the elastoplastic model is evaluated using a classical explicit return-mapping algorithm, see Simo and Hughes [105].

The training data consists of strain as input, uniformly sampled in the interval \( \varepsilon = [0, 0.01] \), and stress as output calculated according to equation (4.6). The yield stress is chosen as \( \sigma_Y = 300 \) MPa, the elastic modulus \( E = 2.1 \times 10^5 \) MPa and the hardening modulus as \( 2.1 \times 10^4 \) MPa. Three datasets are generated, namely, a training set with \( n_{train} = 10240 \) samples and a validation set and test set with \( n_{val} = n_{test} = 1024 \) samples. All three sets are standardized using the mean and standard deviation from the training set. The batch size is chosen as \( n_{batch} = 1024 \). The training is carried out for 500 epochs. The model performing best on the validation set is chosen for subsequent evaluations. The mean relative error accumulated over all time steps and the mean relative error of the last strain step with respect to the test set are reported.

4.2.1. Influence of hyperparameter

The first study investigates the prediction accuracy as a function of (i) the number of output neurons, which participate in the population regression outlined in §2.1 and (ii) different capacities of the
Figure 15. Isotropic hardening—reference solutions. Five stress–strain curves sampled from the isotropic hardening material model for different maximum strains obtained from equation (4.6).

Figure 16. Isotropic hardening—error versus width. The mean relative error of the last strain step versus the number of hidden neurons per layer is shown for different numbers of output neurons in the isotropic hardening experiment from §4 using the SLSTM from equation (4.7).

Figure 17. Isotropic hardening—error versus width. The mean relative error of all strain steps versus the number of hidden neurons per layer is shown for different numbers of output neurons in the isotropic hardening experiment from §4 using the SLSTM from equation (4.7).
SLSTM in the sense of layer width. To this end, the SLSTM defined in equation (4.3) is used, resulting in the following architecture:

\[
S_{\text{SLSTM}}(\varepsilon_t) = h_{t}^{\text{pop}} \circ h_{t}^{\text{dec}} \circ h_{t}^{\text{SLSTM}} \circ h_{t}^{\text{SLSTM}} \circ h_{t}^{\text{SLSTM}} \circ h_{t}^{\text{const}}(\varepsilon_t) = \sigma_t. \tag{4.7}
\]

Multiple simulations with output neurons and hidden layers drawn from the grid \(n_u \times n_o = [16, 32, 64, 128, 256] \times [16, 32, 64, 128, 256]\) are carried out. The resulting mean relative error for all strain steps with respect to the test set is shown in figure 16, whereas the resulting mean relative error of the last strain step with respect to the test set is depicted in figure 17. A clear convergence behaviour can be observed for the number of hidden neurons \(n_u\), where larger numbers of neurons lead to lower errors. For the number of output neurons \(n_o\), a tendency can be observed upon convergence with respect to \(n_u\). For the largest number of hidden neurons \(n_u = 256\), the mean relative error over all strain steps and the mean relative error of the last strain step get larger for \(n_o = [128, 256]\) output neurons, whereas for \(n_o = [16, 32, 64]\) the errors are almost the same. The lowest mean relative error for all strain steps is \(5.2445 \times 10^{-2}\) for \(n_u = 256\) hidden neurons per layer and \(n_o = 64\) output neurons. The lowest mean relative error for the last strain steps is \(2.8729 \times 10^{-3}\) for \(n_u = 256\) hidden neurons per layer and \(n_o = 32\) output neurons. Again, the seemingly high errors are caused by outliers polluting the average, as also observed in §3.2. The same counter-measures discussed in §3.2 can be applied to prohibit outliers, for example, by enforcing thermodynamic consistency.

### 4.2.2. Energy and memory efficiency

For the second experiment, the SLSTM using \(n_o = 64\) output neurons and \(n_u = 256\) hidden neurons per layer are compared with a standard LSTM with an equal number of optimization parameters. The aim of this study is the comparison of the prediction accuracy, but also the difference in memory and energy consumption on neuromorphic hardware. For both ANN variants to be comparable, the same topology is chosen for the LSTM as for the SLSTM, such that

\[
\mathcal{N}_{\text{LSTM}}(\varepsilon_t) = h_{t}^{\text{dense}} \circ h_{t}^{\text{dense}} \circ h_{t}^{\text{LSTM}} \circ h_{t}^{\text{LSTM}} \circ h_{t}^{\text{LSTM}} \circ h_{t}^{\text{const}}(\varepsilon_t) = \sigma_t. \tag{4.8}
\]

where the last two layers are replaced by densely connected conventional feed-forward neural networks. Again, the training was carried out for \(5 \times 10^3\) epochs and the same datasets from the previous experiments are used. The standard LSTM from equation (4.8) reached a mean relative error of \(4.8611 \times 10^{-2}\) over all strain steps and a mean relative error of \(4.7569 \times 10^{-3}\) for the last strain step. The SLSTM from equation (4.7) reached a mean relative error of \(9.3832 \times 10^{-2}\) over all strain steps and a
mean relative error of $4.0497 \times 10^{-3}$ for the last strain step. The resulting prediction for one strain path is illustrated in figure 18. Clearly, both networks are able to accurately predict the history-dependent, nonlinear stress–strain behaviour.

Some deviations from the SLSTM can be seen at the beginning of the curve. The dynamics of the spiking formulation result in a higher mean relative error over all strain steps with respect to the LSTM. However, the endpoint has a better fit than the LSTM. This is seen in the lower error at the last strain step.

To assess the potential of interfacing our model in embedded, resource-constrained sensors in the wild, we performed a series of power profiling experiments for our SNNs (both using LIF neurons and SLSTMs) when processed on the Loihi neuromorphic chip [54]. These results are compared against their non-spiking equivalents on an NVIDIA V100 GPU. Data were extracted using the energy profiler in KerasSpiking v. 0.3.0. The emulation tool takes all non-zero activations, accumulates the operation count of each activation (i.e. the fan-out of the activated neuron) and scales this by the estimated energy cost per operation. Specifically, this operation is a ‘read-and-accumulate’, where the presence of a spike requires all connected synapses to be ‘read’ from memory and accumulated with other terms. Other literature refers to this as ‘SynOps’ or ‘Synaptic Operations’ [106–110]. This is a coarse, but reasonable, approximation for small-scale models as neuromorphic hardware skips processing non-zero activations. For large-scale models that require inter-chip data communication, the energy cost of overhead is not accounted for in such a model. For our purposes, with lightweight, low-power models, this is an acceptable approximation. It is also the de facto metric for measuring energy efficiency in silico from the NeuroBench [110] initiative, which aims to find representative benchmarking of neuromorphic models.

The first difference in energy usage is that the spiking implementation is measured in an ‘event-based’ manner, where processing only occurs when a neuron emits a spike. In contrast, a non-spiking network processed on a GPU continuously computes with all activations. The second difference is that SNNs require multiple strain steps of a forward pass, whereas their non-spiking counterparts do not (unless the input to the network varies over time). Note that the cost of overhead did not need to be accounted for (i.e. transferring data between devices) because all models fit on a single device.

Each network has been broken up into its constituent layers to measure how much they contribute to energy usage on each device. The total energy consumption per forward pass of the non-spiking network on the V100 is 512 nJ, whereas the equivalent SNN is 4.25 nJ. This represents a $120\times$ reduction in energy consumption. The non-spiking LSTM network consumed 5.7 µJ while the proposed SLSTM architecture required 24 nJ, a $238\times$ reduction. Detailed results are summarized in table 1.

### Table 1. Comparison between spiking and non-spiking forward-pass energy consumption and memory usage.

| architecture | energy (Loihi) | energy (GPU) | architecture | energy (Loihi) | energy (GPU) |
|--------------|----------------|--------------|--------------|----------------|--------------|
| dense        |                |              | LSTM         |                |              |
| FC1          | 6.9e-2         | 0.61         | LSTM1        | 0.28           | 2.5          |
| FC2          | 1.3            | 160          | LSTM2        | 1.1            | 2.5e3        |
| FC3          | 1.3            | 160          | LSTM3        | 10             | 2.5e3        |
| FC4          | 0.3            | 160          | FC1          | 1.2            | 2.6          |
| total energy | 4.25           | 512          | total energy | 24             | 5.7e3        |
| reduction    | ×120           |              | reduction factor | ×238         |              |
| synaptic memory | 0.86 MB     |              | synaptic memory | 9.5 MB       |              |

5. Conclusion and outlook

In the present study, a framework for regression using SNNs was proposed based on a membrane potential spiking decoder and a population voting layer. Several numerical examples using different
spiking neural architectures investigated the performance of the introduced topology towards linear, nonlinear and history-dependent regression problems.

First, a simple feed-forward SNN, the LIF, was derived from the classical densely connected feed-forward ANN. It was shown that the SNN can be seen as a special kind of activation function, which produces binary outputs, so-called spikes. These spikes are used to propagate information through a possibly deep SNN. The spikes occur due to the dynamic behaviour of the membrane potential inside the neuron, which rises when spikes appear at the input and decays over time if no spikes appear. If a certain threshold value is reached, the membrane potential is reset and the neuron emits a spike itself. This formulation introduces more hyperparameters, which fortunately can be learned during training. The spikes introduce sparsity in the network, which can be effectively exploited by neuromorphic hardware to improve latency, power and memory efficiency. The non-differentiability of the binary spikes is circumvented by surrogate gradients during backpropagation.

Next, a network topology was proposed, which decodes binary spikes into real numbers, which is essential for all kinds of regression problems. A decoding layer takes the membrane potentials of all neurons in the last spiking layer and propagates them to a population voting layer, which provides its mean potential resulting in a real number. The proposed topology can be used for arbitrary temporal input and output dimensions. A simple experiment on a linear elastic material model using LIFs showed that the proposed topology is able to regress the problem. It was shown that errors are introduced for a large number of strain steps. This problem was overcome by introducing RLIF, which extends the LIF by recurrent feedback loops. An experiment using a nonlinear Ramberg–Osgood plasticity model showed that the proposed topology using RLIF is able to regress varying yield limits accurately. The final extension was concerned with the introduction of explicit long-term memories inspired by the classical LSTM formulation, resulting in a spiking LSTM. The performance of this SLSTM was investigated on a history-dependent isotropic hardening model, where different load paths were accurately regressed. During prediction, the SLSTM was able to generalize even better than the LSTM for the final load step. Furthermore, the convergence of the proposed method was shown. Note that an extension towards two- and three-dimensional mechanical problems is equally possible. This will involve learning a functional relationship for each stress component, which can be either achieved by a multi-input/output architecture or by using individual networks for each stress component. This treatment is analogous to second-generation ANN, see for instance [111], where stress data has been decoupled using Proper orthogonal decomposition (POD).

Power profiling and memory analysis were conducted on the LIF and SLSTM networks to compare efficiency on neuromorphic hardware as against a GPU. The Loihi neuromorphic processor was able to achieve a 120× reduction in energy consumption when processing the dense LIF network, and the SLSTM offered a 238× reduction in energy during inference.

The range of possible future application scenarios enabled by regression with SNN are manifold. For instance, today’s sensing systems cannot capture all quantities that are relevant for structural health monitoring. In the context of mechanics, displacement and strain are quite easy to assess, but the mechanical stress, which reflects the actual response of structures and materials to deformation, remains a so-called hidden quantity. Physics-informed machine learning offers the potential to reconstruct hidden quantities from data by leveraging information from physical models, given in the form of partial differential equations. It is expected that the developments in the field of neuromorphic hardware will foster the development of a new generation of embedded systems, which will ultimately enable control of structures and processes based on partial differential equations.

**Ethics.** This work did not require ethical approval from a human subject or animal welfare committee.

**Data accessibility.** The code is available at: https://github.com/ahenkes1/HENKES_SNN and Zenodo [112].

**Declaration of AI use.** We have not used AI-assisted technologies in creating this article.

**Authors’ contributions.** A.H.: conceptualization, data curation, formal analysis, funding acquisition, investigation, methodology, project administration, resources, software, supervision, validation, visualization, writing—original draft, writing—review and editing; J.K.E.: conceptualization, funding acquisition, software, supervision, writing—original draft, writing—review and editing; H.W.: conceptualization, funding acquisition, resources, supervision, visualization, writing—review and editing.

All authors gave final approval for publication and agreed to be held accountable for the work performed therein.
Conflict of interest declaration. The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Funding. No funding has been received for this article.

Acknowledgements. A.H. was supported by an ETH Zurich Postdoctoral Fellowship. H.W. and A.H. acknowledge the support of the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) in the project DFG 25504259/GRK2075-2: Modelling the constitutional evolution of building materials and structures with respect to aging. H.W. additionally acknowledges funding in the project DFG 501798687: Monitoring data-driven life cycle management with AR based on adaptive, AI-supported corrosion prediction for reinforced concrete structures.

References

1. Hornik K, Stinchcombe M, White H. 1989 Multilayer feedforward networks are universal approximators. Neural Netw. 2, 359–366. (doi:10.1016/0893-6080(89)90020-8)
2. Cybenko G. 1989 Approximation by superpositions of a sigmoidal function. Math. Control. Signal. Syst. 2, 303–314. (doi:10.1007/BF02551274)
3. Chen T, Chen H. 1995 Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. IEEE Trans. Neural Netw. 6, 911–917. (doi:10.1109/72.392253)
4. Julius B, Philipp G, Gitta K, Philipp P. 2021 The modern mathematics of deep learning. arXiv. See https://arxiv.org/abs/2105.04026
5. Kutz JN. 2017 Deep learning in fluid dynamics. J. Fluid Mech. 814, 1–4. (doi:10.1017/jfm.2016.803)
6. Wei-Wei Z, Bernd RN. 2021 Artificial intelligence in fluid mechanics. Acta Mech. Sin. 37, 1715–1717. (doi:10.1007/s10409-021-01154-3)
7. Cai S, Mao Z, Wang Z, Yin M, Karniadakis GE. 2021 Physics-informed neural networks (PINNs) for fluid mechanics: a review. Acta Mech. Sin. 37, 1–12. (doi:10.1007/s10409-021-01148-1)
8. Raissi M, Yazdani A, Karniadakis GE. 2020 Hidden fluid mechanics: learning velocity and pressure fields from flow visualizations. Science 367, 1026–1030. (doi:10.1126/science.aaw4741)
9. Henning W, Christian W, Peter W. 2020 The neural particle method – an updated Lagrangian physics informed neural network for computational fluid dynamics. Comput. Methods Appl. Mech. Eng. 368, 113127. (doi:10.1016/j.cma.2020.113127)
10. Haghighat E, Raissi M, Moure A, Gomez H, Juanes R. 2021 A physics-informed deep learning framework for inversion and surrogate modeling in solid mechanics. Comput. Methods Appl. Mech. Eng. 379, 113741. (doi:10.1016/j.cma.2021.113741)
11. Buffa G, Fratini L, Micari F. 2012 Mechanical and microstructural properties prediction by artificial neural networks in FSW processes of dual phase titanium alloys. J. Manuf. Process. 14, 289–296. (doi:10.1016/j.jmapro.2011.10.007)
12. Abueidda DW, Lu Q, Koric S. 2021 Meshless physics-informed deep learning method for three-dimensional solid mechanics. Numer. Meth. Eng. 122, 7182–7201. (doi:10.1002/nme.6828)
13. Nie Z, Jiang H, Kara LB. 2020 Stress field prediction in cantilevered structures using convolutional neural networks. J. Comput. Inf. Sci. Eng. 20, 011102. (doi:10.1115/1.4044097)
14. Henkes A, Caylak I, Mahnken R. 2021 A deep learning driven pseudospectral PCE based FFT homogenization algorithm for complex microstructures. Comput. Methods Appl. Mech. Eng. 385, 114070. (doi:10.1016/j.cma.2021.114070)
15. Henkes A, Wessels H, Mahnken R. 2022 Physics informed neural networks for continuum micromechanics. Comput. Methods Appl. Mech. Eng. 393, 114790. (doi:10.1016/j.cma.2022.114790)
16. Miannoodi JR, H. Siboni N, Raabe D. Teaching solid mechanics to artificial intelligence—a fast solver for heterogeneous materials. NPJ Comput. Mater. 7, 99. (doi:10.1038/s41524-021-00571-z)
17. Méndez-Mancilla A, Wessels HH, Legut M, Kadina A, Mabuchi M, Walker J, Robb GB, Holden K, Sanjana NE. 2022 Chemically modified guide Rnas enhance CRISPR-Cas13 knockdown in human cells. In Current trends and open problems in computational mechanics, pp. 569–579. Springer. See https://www.sciencedirect.com/science/article/pii/S2451945621003512.
18. Dehghani H, Zilian A. 2020 Poroeelastic model parameter identification using artificial neural networks: on the effects of heterogeneous porosity and solid matrix Poisson ratio. Com. Math. Mech. 66, 625–649. (doi:10.1007/s00466-020-1866-4)
19. Thakolkaran P, Joshi A, Zhong W, Fischel M, De Lorenzis L, Kumar S. 2022 NN-EUCLID: Deep-learning hyperelasticity without stress data. arXiv 169, 105076. (https://arxiv.org/abs/2205.06664)
20. Zhang E, Yin M, Karniadakis GE. 2020 Physics-informed neural networks for nonhomogeneous material identification in elasticity imaging. arXiv. See https://arxiv.org/abs/2009.04525
21. Zhang E, Yao M, Karniadakis GE, Suresh S. 2022 Analyses of internal structures and defects in materials using physics-informed neural networks. Sci. Adv. 8, eabk0644. (doi:10.1126/sciadv.abk0644)
22. Anton D, Wessels H. 2022 Physics-informed neural networks for material model calibration from full-field displacement data. arXiv. See https://arxiv.org/pdf/2212.07723.
23. As’ad F, Avery P, Farhat C. 2022 A mechanics-informed artificial neural network approach in data-driven constitutive modeling. J. Numer. Meth. Eng. 123, 2738–2759. (doi:10.1002/nme.6957)
Yang H, Qiu H, Xiang Q, Tang S, Guo X. 2020 Exploring elastoplastic constitutive law of microstructured materials through artificial neural network—a mechanism-based data-driven approach. J. Appl. Mech. 87, 091005. (doi:10.1115/1.4047208)

Xu K, Huang DZ, Darve E. 2021 Learning constitutive relations using symmetric positive definite neural networks. J. Comput. Phys. 428, 110072. (doi:10.1016/j.jcp.2020.110072)

Fernández M, Rezaei S, Rezaei Mianroodi J, Fritzen F, Reese S. 2020 Application of artificial neural networks for the prediction of interface mechanics: a study on grain boundary constitutive behavior. Adv. Model. Simul. in Eng. Sci. 7, 1–27. (doi:10.1186/s40323-019-0138-7)

Aldakheel F, Salem R, Wriggers P. 2021 Feed-forward neural networks for failure mechanics problems. Appl. Sci. 11, 6483. (doi:10.3390/app11146483)

Liu SW, Huang JH, Sung JC, Lee CC. 2002 Detection of cracks using neural networks and computational mechanics. Comput. Methods Appl. Mech. Eng. 191, 2831–2845. (doi:10.1016/S0045-7825(02)00221-9)

Henkes A, Wessels H. 2022 Three-dimensional microstructure generation using generative adversarial neural networks in the context of continuum micromechanics. arXiv. See https://arxiv.org/abs/2206.01693

Hsu T, Eppling WK, Kim H, Abernathy HW, Hackett GA, Rollett AD, Salvador PA, Holm EA. 2021 Microstructure generation via generative adversarial network for heterogeneous, topologically complex 3D materials. JOM 73, 90–102. (doi:10.1007/s11837-020-04484-y)

Mosser L, Dubrule O, Blunt MJ. 2017 Reconstruction of three-dimensional porous media using generative adversarial neural networks. Phys. Rev. E 96, 043309. (doi:10.1103/PhysRevE.96.043309)

Ma J, Dong S, Chen G, Peng P, Qian L. 2021 A data-driven normal contact force model based on artificial neural network for complex contacting surfaces. Mech. Syst. Signal Process. 156, 107612. (doi:10.1016/j.ymssp.2021.107612)

Öner E, Şengül Şahanoğlu B, Uzun Yavuza E, Adiyaman G, Yavọla M, Birinci A. 2022 On the plane receding contact between two functionally graded layers using computational, finite element and artificial neural network methods. Z. Angew. Math. Mech. 102, e202100287. (doi:10.1002/zamm.202100287)

Ardestani MM, Chen Z, Wang L, Lian Q, Liu Y, He J, Li D, Jin Z. 2014 Feed forward artificial neural network to predict contact force at medial knee joint: application to gait modification. Neurocomputing 139, 114–129. (doi:10.1016/j.neucom.2014.02.054)

Feng Z, Yan J, Gao Y. 2022 Prediction of contact resistance between copper blocks under cyclic load based on deep learning algorithm. AIP Adv. 12, 075009. (doi:10.1063/5.0095871)

Cai S, Wang Z, Wang S, Perdikaris P, Karniadakis GE. 2021 Physics-informed neural networks for heat transfer problems. J. Heat Transfer. 143, 6. (doi:10.1115/1.4050542)

Laubscher R. 2021 Simulation of multi-species flow and heat transfer using physics-informed neural networks. Phys. Fluids 33. (doi:10.1063/5.0058529)

Tamaddon-Jahromi HR, Chakshu NK, Sazanov I, Evans LM, Thomas H, Nithiarasu P. 2020 Data-driven inverse modelling through neural network (deep learning) and computational heat transfer. Comput. Methods Appl. Mech. Eng. 369, 113217. (doi:10.1016/j.cma.2020.113217)

Niaki SA, Haghighat E, Campbell T, Poursartip A, Vaziri R. 2021 Physics-informed neural network for modelling the thermochemical curing process of composite-tool systems during manufacture. Comput. Methods Appl. Mech. Eng. 384, 113959. (doi:10.1016/j.cma.2021.113959)

Ramudo VAF. 2020 Machine learning to build reduced order models of solid mechanic models with uncertainty. Master’s thesis, Universitat Politècnica de Catalunya.

Balokas G, Czichon S, Rolfes R. 2018 Neural network assisted multiscale analysis for the elastic properties prediction of 3D braided composites. Front. Mech. Eng. 6, 110. (doi:10.3389/fmats.2019.00110)

Kumar S, Kochmann DM. 2021 What machine learning can do for computational solid mechanics. In (eds F Aldakheel, B Hudobivnik, M Soleimani, H Wessels, C Weißenfels, M Marino), pp. 275–285. Cham: Springer. See https://link.springer.com/chapter/10.1007/978-3-030-87312-7_27. (doi:10.1007/978-3-030-87312-7_27)

Bleckschmidt J, Ernst OG. 2021 Three ways to solve partial differential equations with neural networks — A review. GAMM-Mitteilungen 44, e20210006. (doi:10.1002/gamm.20210006)

Roy K, Jaiswal A, Panda P. 2019 Towards spike-based machine intelligence with neuromorphic computing. Nature 575, 607–617. (doi:10.1038/s41586-019-1677-2)

Indiveri G, Liu SC. 2015 Memory and Information Processing in Neuromorphic Systems. Proc. IEEE 103, 1379–1397. (doi:10.1109/JPROC.2015.244094)

Burr GW et al. 2017 Neuromorphic computing using non-volatile memory. Adv. Phys.-X 2, 89–124. (doi:10.1080/23746149.2016.1259585)

Narouie VB, Wessels H, Römer U. 2023 Inferring displacement fields from sparse measurements using the statistical finite element method. Mech. Syst. Signal Process 200, 110574. (doi:10.1016/j.ymssp.2023.110574)

Perez-Nieves N, Goodman D. 2021 Sparse Spiking gradient descent. Adv. Neural Inf. Process. Syst 34, 1795–1808. (doi:10.1016/j.ymssp.2023.110574)
52. Olshausen BA, Field DJ. 2006 What is the other 85 percent of V1 doing. In 23 problems in systems neuroscience (eds L van Hemmen, T Sejnowski), pp. 182–211, vol. 23. See https://global.oup.com/academic/product/23-problems-in-systems-neuroscience-9780195148220?cc=ch&lang=en&s; (doi:10.1093/acprof:oso/9780195148220.001.0001)

53. Gerstner W, Kistler WM. 2002 Spiking neuron models: single neurons, populations, plasticity. Cambridge, UK: Cambridge University Press. (doi: 10.1017/CBO9780511815706)

54. Davies M et al. 2018 Loihi: a neuromorphic manycore processor with on-chip learning. IEEE Micro 38, 82–99. (doi:10.1109/MM.2018.112130359)

55. Furber SB, Galluppi F, Temple S, Plana LA. 2014 The SpiNNaker project. Proc. IEEE 102, 652–665. (doi:10.1109/PROC.2014.2304638)

56. Merolla PA et al. 2014 Artificial brains: a million spiking-neuron integrated circuit with a scalable communication network and interface. Science 345, 668–673. (doi:10.1126/science.1254642)

57. Azghadi MR, Lamme C, Eshraghian JK, Payvand M, Donati E, Linares-Barranco B, Indiveri G. 2020 Hardware implementation of deep network accelerators towards healthcare and biomedical applications. IEEE Trans. Biomed. Circuits Syst. 14, 1138–1159. (doi:10.1109/TBCAS.2020.3036081)

58. Frenkel C, Indiveri G. 2022 ReckOn: a 28nm sub-mm² task-agnostic spiking recurrent neural network processor enabling on-chip learning over second-long timescales. In 2022 IEEE International Solid-State Circuits Conference (ISSCC), vol. 65. pp. 1–3 IEEE.

59. Cecolini E, Frenkel C, Shrestha SB, Taverni G, Khezeli F, Payvand M, Donati E. 2020 Hand-gesture recognition based on EMG and event-based camera sensor fusion: a benchmark in neuromorphic computing. Front. Neurosci. 14, 637. (doi:10.3389/fnins.2020.00637)

60. Orchard G, Frady EP, Rubin DBD, Sanborn S, Shrestha SB, Sommer FT, Davies M. 2021 Efficient Neuromorphic signal processing with Loihi 2. In 2021 IEEE Workshop on Signal Processing Systems (SiPS) pp. 254–259. IEEE.

61. Yang S, Tan J, Chen B. 2022 Robust spike-based continual meta-learning improved by restricted minimum error entropy criterion. Entropy 24, 455. (doi:10.3390/e24040455)

62. Yang S, Chen B. 2023 Effective surrogate gradient learning with high-order information bottleneck for spike-based machine intelligence. IEEE Trans. Neural Netw. Learn. Syst. 1–15. (doi:10.1109/TNNLS.2023.3329252)

63. Yang S, Wang H, Chen B. 2023 SIBoLS: robust and energy-efficient learning for spike based machine intelligence framework. IEEE Trans. Cogn. Dev. Syst. 1–13. (doi:10.1109/TCD.2023.3329532)

64. Yang S, Chen B. 2023 Snib: Improving spike-based machine learning using nonlinear information bottleneck. IEEE Trans. Syst. Man Cybern. Syst. 53, 7852–7863. (doi:10.1109/TSMC.2023.3300318)

65. Abadi M, et al. 2015 TensorFlow: large-scale machine learning on heterogeneous systems. See https://www.tensorflow.org/

66. Paszke A, et al. 2019 Pytorch: an imperative style, high-performance deep learning library. In Advances in neural information processing systems 32 (eds H Wallach, H Larochelle, A Beygelzimer, F d’Alché-Buc, E Fox, R Garnett), pp. 8024–8035. Curran Associates, Inc. See https://proceedings.neurips.cc/paper/2019/hash/ebdca288fe79f226af97012727740-Abstract.html.

67. Eshraghian JK, Ward M, Neftci E, Wang X, Lenz G, Dwivedi G, Bennamoun M, Jeong DS, Lu WD. 2021 Training spiking neural networks using lessons from deep learning. arXiv. See https://arxiv.org/abs/2109.12894

68. Fang W, Yu Z, Chen Y, Huang T, Masquelier T, Tian Y. 2021 Deep residual learning in spiking neural networks. Adv. Neural Inf. Process. Syst. 34, 21. (doi:10.48550/arXiv.2102.04159)

69. Bellec G, Dalaj D, Subramoney A, Legenstein R, Maass W. 2018 Long short-term memory and learning-to-learn in networks of spiking neurons. Adv. Neural Inf. Process. Syst. 31. (doi:10.48550/arXiv.1803.09574)

70. Rao A, Plank P, Wild A, Maass W. 2022 A long short-term memory for AI applications in spike-based neuromorphic hardware. Nat. Mach. Intell. 4, 467–479. (doi:10.1038/s42256-022-00480-w)

71. Yang Y, Eshraghian J, Truong ND, Nikpour A, Kavehei O. 2022 Neuromorphic deep spiking neural networks for seizure detection. See https://www.techxrv.org/

72. Patel K, Hunsberger E, Batir S, Eliasmith C. 2021 A spiking neural network for image segmentation. arXiv. See https://arxiv.org/abs/2106.08921

73. Barchid S, Mennesson J, Eshraghian J, Djéraba C, Bennamoun M. 2023 Spiking neural networks for frame-based and event-based single object localization. arXiv. See https://arxiv.org/abs/2206.06506

74. Moro F et al. 2022 Neuromorphic object localization using resistive memories and ultrasonic transducers. Nat. Commun. 13, 3506. (doi:10.1038/s41467-022-31157-y)

75. Iannella N, Back AD. 2001 A spiking neural network architecture for nonlinear function approximation. Neural Netw. 14, 933–939. (doi:10.1016/S0893-6080(01)00080-6)

76. Gehrig M, Shrestha SB, Mouritzen D, Scaramuzza D. 2020 Event-based angular velocity regression with spiking networks. In 2020 IEEE Int. Conf. on Robotics and Automation (ICRA), pp. 4195–4202 IEEE.

77. Rançon U, Cuadrado-Anibarro J, Cottereau BR, Masquelier T. 2021 Stereospike: depth learning with a spiking neural network. arXiv. See https://arxiv.org/abs/2109.13751

78. Kahana A, Zhang Q, Gleyzer L, Karniadakis GE. 2022 Spiking neural operators for scientific machine learning. arXiv. See https://arxiv.org/abs/2205.10130

79. Lu L, Jin P, Pang G, Zhang Z, Karniadakis GE. 2021 Learning nonlinear operators via deepONet based on the universal approximation theorem of operators. Nat. Mach. Intell. 3, 218–229. (doi:10.1038/s42256-021-00302-5)

80. Shrestha SB, Orchard G. 2018 Slayer: spike layer error reassignment in time. Adv. Neural Inf. Process. Syst. 31. (doi:10.48550/arXiv.1810.08646)
